МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра дискретной математики и алгоритмики

КОСТЮКОВИЧ
Полина Кирилловна

МЕТОДЫ РЕШЕНИЯ ЗАДАЧИ О ПЛАНИРОВАНИИ РЕЙСОВ ВОЗДУШНЫХ СУДОВ

Дипломная работа

Научный руководитель:
кандидат физико-математических наук
В.В. Мушко

Допущена к защите

«19» мая 2025 г.

Заведующий кафедрой дискретной математики и алгоритмики
доктор физико-математических наук, профессор В.М. Котов

Минск, 2025

ОГЛАВЛЕНИЕ

РЕФЕРАТ

Дипломная работа: 60 страниц, 6 иллюстраций, 1 таблица, 18 источников.

Ключевые слова: ЗАДАЧА КОМБИНАТОРНОЙ ОПТИМИЗАЦИИ, МАТЕМАТИЧЕСКАЯ МОДЕЛЬ, ЦЕЛОЧИСЛЕННОЕ ЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ, МЕТАЭВРИСТИКА, МЕТОД ИМИТАЦИИ ОТЖИГА, ПРОГРАММИРОВАНИЕ В ОГРАНИЧЕНИЯХ.

Объект исследования: задача составления расписаний.

Предмет исследования: задача о планировании рейсов воздушных судов.

Цель работы: разработать и реализовать практически применимое решение задачи о планировании рейсов воздушных судов.

Методы исследования: методы теории алгоритмов, методы и алгоритмы решения задач комбинаторной оптимизации.

Полученные результаты и их новизна: построены и программно реализованы три математические модели задачи о планировании рейсов воздушных судов: в терминах целочисленного линейного программирования, в терминах задачи комбинаторной оптимизации и в терминах задачи программирования в ограничениях. Описаны два алгоритма генерации начальных расписаний, сгенерированы тестовые начальные расписания различных размеров. Проведены вычислительные эксперименты, на основании которых выполнен анализ применимости и эффективности предложенных решений.

Достоверность материалов и результатов дипломной работы: использованные материалы и результаты дипломной работы являются достоверными. Работа выполнена самостоятельно.

Область возможного практического применения: результаты работы могут быть использованы авиакомпаниями для составления расписаний рейсов воздушных судов.

РЭФЕРАТ

Дыпломная праца: 60 старонак, 6 ілюстрацый, 1 табліца, 18 крыніц.

Ключавыя словы: ЗАДАЧА КАМБІНАТОРНАЙ АПТЫМІЗАЦЫІ, МАТЭМАТЫЧНАЯ МАДЭЛЬ, ЦЭЛАЛІКАВАЕ ЛІНЕЙНАЕ ПРАГРАМАВАННЕ, МЕТАЭЎРЫСТЫКА, МЕТАД ІМІТАЦЫІ АДПАЛУ, ПРАГРАМАВАННЕ Ў АБМЕЖАВАННЯХ.

Аб'ект даследавання: задача складання раскладаў.

Прадмет даследавання: задача аб планаванні рэйсаў паветраных суднаў.

Мэта даследавання: распрацаваць і рэалізаваць практычна прыдатнае рашэнне задачы аб планаванні рэйсаў паветраных суднаў.

Метады даследавання: метады тэорыі алгарытмаў, метады і алгарытмы рашэння задач камбінаторнай аптымізацыі.

Атрыманыя вынікі і іх навізна: пабудаваны і праграмна рэалізаваны тры матэматычныя мадэлі задачы аб планаванні рэйсаў паветраных суднаў: у тэрмінах цэлалікавага лінейнага праграмавання, у тэрмінах задачы камбінаторнай аптымізацыі і ў тэрмінах задачы праграмавання ў абмежаваннях. Апісаны два алгарытма генерацыі пачатковых раскладаў, згенераваны тэставыя пачатковыя расклады розных памераў. Праведзены вылічальныя эксперыменты, на падставе якіх выкананы аналіз прыдатнасці і эфектыўнасці прапанаваных рашэнняў.

Даставернасць матэрыялаў і вынікаў дыпломнай працы: выкарыстаныя матэрыялы і вынікі дыпломнай Працы з'яўляюцца даставернымі. Праца выканана самастойна.

Вобласць магчымага практычнага прымянення: вынікі працы могуць быць выкарыстаныя авіякампаніямі для складання раскладаў рэйсаў паветраных суднаў.

ABSTRACT

Diploma work: 60 pages, 6 illustrations, 1 table, 18 sources.

Keywords: COMBINATORIAL OPTIMIZATION PROBLEM, MATHEMATICAL MODEL, INTEGER LINEAR PROGRAMMING, METAHEURISTIC, SIMULATED ANNEALING, CONSTRAINT PROGRAMMING.

The object of the research is the scheduling problem.

The subject of the research is the aircraft flight scheduling problem.

The purpose of the research is to develop and implement a practically applicable solution for the aircraft flight scheduling problem.

Methods of research: theory of algorithms methods, methods and algorithms for solving combinatorial optimization problems.

The results of the work and their novelty. Three mathematical models of the aircraft flight scheduling problem were developed and implemented: in terms of integer linear programming, combinatorial optimization, and constraint programming. Two algorithms for initial schedule generation were described, and test initial schedules of various sizes were generated. Computational experiments were conducted, and based on their results, the applicability and efficiency of the proposed solutions were analyzed.

Authenticity of the materials and results of the diploma work. The materials used and the results of the diploma work are authentic. The work has been put through independently.

Recommendations on the usage. The results of the work can be used by airlines for flight scheduling.

ВВЕДЕНИЕ

В современном мире индустрия авиаперевозок является важной отраслью с высокими затратами и довольно низкой рентабельностью, поэтому авиакомпании стремятся тщательно планировать свои расписания, чтобы обеспечить максимальную прибыль. Задачи планирования, с которыми сталкивается авиационная отрасль, гораздо более сложны, чем традиционные задачи планирования загрузки устройств. Задачи планирования загрузки устройств в большинстве своем имеют дело с упорядочиванием и планированием набора работ, которые должны быть выполнены на одном или нескольких устройствах. С другой стороны, авиакомпании сталкиваются с рядом взаимосвязанных сложных ограничений. Они должны назначить различные воздушные суда тысячам рейсов, учитывая различные условия связности и совместимости и требования к техническому обслуживанию судов; в то же время необходимо назначить экипажи на различные рейсы таким образом, чтобы не только выполнялись условия связности и совместимости, но и соблюдались санитарные требования для членов экипажа.

Задача планирования расписания включает в себя множество различных подзадач, таких как планирование полетов воздушных судов, планирование работы экипажа, управление сбоями в расписании и многие другие задачи длительного планирования. Признавая центральную роль расписания, специалисты по исследованию операций как внутри авиакомпаний, так и за их пределами работают над разработкой методов получения оптимальных расписаний уже с 1950-х годов.

Цель данной дипломной работы — разработать и реализовать практически применимое решение задачи о планировании рейсов воздушных судов.

Для достижения цели работы были поставлены следующие задачи:

- построить различные математические модели задачи;
- программно реализовать построенные модели;
- разработать и программно реализовать алгоритм генерации тестовых данных;
- провести вычислительный эксперимент и проанализировать результаты.

В рамках данной дипломной работы рассмотрена задача назначения воздушных судов рейсам заранее известного расписания. Представлены и программно реализованы три математические модели задачи: в терминах целочисленного линейного программирования, в терминах задачи комбинаторной оптимизации и в терминах задачи программирования в ограничениях. Также описаны два алгоритма генерации начальных расписаний, сгенерированы тестовые начальные расписания различных размеров. Проведены вычислительные эксперименты, на основании которых выполнен анализ применимости и эффективности предложенных решений.

ГЛАВА 1

КРАТКИЕ СВЕДЕНИЯ ОБ ИСПОЛЬЗОВАННЫХ МЕТОДАХ РЕШЕНИЯ ЗАДАЧИ

1.1 Задача целочисленного линейного программирования

1.1.1 Определения и постановка задачи

Определение 1. Задачей целочисленного линейного программирования (далее ЦЛП) называется задача вида:

$$\begin{cases} c^{\mathrm{T}}x \to \min \\ Ax = b \\ d_1 \leqslant x \leqslant d_2 \\ x \in \mathbb{Z}^n \end{cases} \tag{1.1}$$

где $x \in \mathbb{R}^n$ — вектор переменных, $c \in \mathbb{R}^n$ — вектор стоимостей, $A \in \mathbb{R}^{m \times n}$ — матрица ограничений, $d_1, d_2 \in \mathbb{R}^n \cup \{+\infty, -\infty\}$ — нижние и верхние границы переменных соответственно.

Определение 2. Вектор $x \in \mathbb{R}^n$, удовлетворяющий ограничениям задачи (1.1), называется планом этой задачи.

Определение 3. План задачи (1.1), на котором значение $c^{\mathrm{T}}x$ минимально, называется оптимальным планом этой задачи.

Определение 4. Релаксацией задачи ЦЛП вида (1.1) называется задача вида

$$\begin{cases} c^{\mathrm{T}}x \to \min \\ Ax = b \\ d_1 \leqslant x \leqslant d_2 \end{cases} \tag{1.2}$$

т.е. это задача (1.1), в которой отсутствует условие целочисленности вектора x.

Доказано, что в общем случае задача целочисленного линейного программирования является NP-трудной [8, с. 232-235].

1.1.2 Метод ветвей и границ для задачи ЦЛП

Одним из наиболее часто применяемых методов решения задачи ЦЛП является метод ветвей и границ. Идея этого метода состоит в такой организации перебора планов задачи (1.1), при которой удается отбрасывать не отдельные точки, а достаточно большие множества заведомо неоптимальных планов [17, с. 295].

Запишем задачу (1.1) в виде:

$$Z = \max\{c^{\mathrm{T}}x : x \in S\} \tag{1.3}$$

Суть метода состоит в том, чтобы разбить задачу на несколько подзадач меньшего размера, которые проще решить, а потом из этих решений составить решение исходной задачи. Подзадачи — это задачи вида (1.3) с множеством планов, которое является подмножеством множества планов исходной задачи. Классическим способом представить такой подход является дерево перебора [12, с. 113]. Корню этого дерева соответствует исходная задача, а детям каждой вершины — подзадачи, на которые разбивается задача, соответствующая этой вершине.

Чтобы избежать полного перебора, каждой вершине дерева приписывается нижняя оценка для значения целевой функции на множестве планов подзадачи, соответствующей этой вершине. Также поддерживается значение рекорда — минимальное значение целевой функции, для которого уже найден целочисленный план (изначально рекорд равен $+\infty$). Если какой-то вершине дерева перебора приписана нижняя оценка, которая больше текущего рекорда, то решать подзадачу, соответствующую этой вершине, не имеет смысла, поэтому данная ветвь отсекается по рекорду.

Более формально алгоритм наиболее распространенного варианта метода ветвей и границ для задачи ЦЛП можно описать так:

1. Пусть H — список еще не решенных подзадач вида (1.3), r — минимальное значение $c^{\mathrm{T}}x$, для которого уже найден целочисленный план. Изначально H состоит из исходной задачи, $r = +\infty$.

2. Если $H = \varnothing$, $Z = r$, задача решена.

3. Выбирается $Q \in H$, Q исключается из H.

4. Решается релаксация задачи Q. Если релаксация не имеет решений, задаче Q приписывается нижняя оценка, равная $+\infty$. Иначе пусть $\overline{x}$ — оптималь-

ный план релаксации.

5. Вершине дерева перебора, соответствующей задаче Q, приписывается нижняя оценка, равная $c^{\mathrm{T}}\overline{x}$. Если эта нижняя оценка не меньше r, то ветвь, соответствующая задаче Q, отсекается по рекорду.

6. Если $\overline{x} \in \mathbb{Z}^n$, обновляется значение r. Дальше эта ветвь не рассматривается, т.к. решение задачи Q уже найдено.

7. Пусть $\overline{x}_i \notin \mathbb{Z}$. Тогда множество S' планов задачи Q разбивается на множества $S_1 = S' \cap \{x_i \leqslant \lfloor \overline{x}_i \rfloor\},\ S_2 = S' \cap \{x_i \geqslant \lceil \overline{x}_i \rceil\}$, после чего задачи вида (1.3) с множествами планов S_1 и S_2 добавляются в H.

В методе ветвей и границ важную роль играет стратегия выбора подзадачи из списка еще не решенных подзадач или схема ветвления. Существует несколько таких схем, например, схема полного ветвления и схема одностороннего ветвления. В соответствии с первой стратегией из списка подзадач выбирается подзадача с наименьшей нижней оценкой, что гарантирует оптимальность первого найденного целочисленного плана. Второй метод состоит в том, что на каждом шаге подзадача выбирается только среди подзадач, полученных разбиением задачи на предыдущем шаге [17, с. 296]. Такой подход позволяет быстро найти какой-нибудь целочисленный план и обновить значение рекорда, что впоследствии позволит производить более эффективные отсечения.

1.1.3 Этапы решения задачи ЦЛП

В настоящее время существует несколько программ и модулей, называемых решателями ЦЛП, например, HiGHS, SCIP (открытые), Gurobi (коммерческий). В каждом из них реализован многоступенчатый процесс решения задачи ЦЛП, который может включать следующие этапы:

- **Предобработка.** Предобработка — это способ преобразовать исходную задачу в эквивалентную ей, причем полученная задача решается проще, чем исходная. Поскольку большинство задач ЦЛП на практике содержат избыточную информацию, что сказывается на скорости решения, во всех конкурентоспособных решателях в том или ином виде присутствует этап предобработки. Он может заключаться, например, в удалении избыточных ограничений и переменных, которые, исходя из ограничений, могут принимать единственное значение [1, с. 133].

- **Поиск первичной эвристики.** Цель этого этапа — найти какой-нибудь

(желательно как можно более близкий к оптимальному) план задачи (1.1) за сравнительно короткое время. Если удается найти такой план, то при дальнейшем применении метода ветвей и границ сразу можно отсекать по рекорду ветви с большей нижней границей (в классическом методе ветвей и границ начальный рекорд равен $+\infty$). Это может значительно уменьшить дерево перебора и, следовательно, ускорить поиск решения [1, с. 117].

- **Решение преобразованной задачи.** Обычно для решения задачи применяется описанный выше метод ветвей и границ.

В целом, современные решатели для большинства задач ЦЛП находят решение довольно быстро. Поскольку многие производственные задачи, задачи планирования и оптимизации можно свести к задаче ЦЛП, поиск решения задачи ЦЛП за приемлемое время играет важную роль в различных сферах. Поэтому решатели ЦЛП и по сей день продолжают активно совершенствоваться.

1.2 Метаэвристические алгоритмы для задач комбинаторной оптимизации

1.2.1 Определения и постановка задачи комбинаторной оптимизации

Многие оптимизационные задачи включают в себя поиск наилучшей в некотором смысле конфигурации множества переменных для достижения каких-то целей. Такие задачи естественным образом делятся на 2 категории: те, в которых решения выражены с помощью переменных, принимающих вещественные значения, и те, в которых решения представляются переменными, принимающими дискретные значения. Последние включают в себя класс задач, которые называются задачами комбинаторной оптимизации (далее КО). Цель задачи КО — найти объект с какими-то свойствами из конечного или счетного множества объектов. Обычно таким объектом является целое число, подмножество, перестановка или графовая структура.

Определение 5. Задача комбинаторной оптимизации $P = (S, f)$ может быть определена следующими объектами:

- множеством переменных $X = \{x_1, ..., x_n\}$;
- областями определения переменных $D_1, ..., D_n$;
- ограничениями на переменные;

- целевой функцией f, которую нужно минимизировать, где
$f : D_1 \times ... \times D_n \to \mathbb{R}^+$.

Множеством всевозможных допустимых назначений переменных является

$$S = \{s = \{(x_1, v_1), ..., (x_n, v_n)\} | v_i \in D_i, s \text{ удовлетворяет всем ограничениям}\} \tag{1.4}$$

S обычно называют пространством поиска (или пространством решений), поскольку каждый элемент этого множества потенциально может быть решением задачи. Решить задачу комбинаторной оптимизации — значит найти решение $s^* \in S$ с минимальным значением целевой функции, то есть $f(s^*) \leqslant f(s)$ $\forall s \in S$. s^* называется глобальным оптимальным решением задачи (S, f).

Примерами задач КО могут служить задача коммивояжёра, задачи планирования и составления расписаний. Из-за практической важности задач КО было разработано множество алгоритмов их решения. Эти алгоритмы можно разделить на точные и приближенные. Для точных алгоритмов гарантируется, что для любой задачи КО конечного размера (т.е. с конечным пространством решений) оптимальное решение будет найдено за ограниченное время. Однако поскольку задачи КО являются NP-трудными, для них не существует полиномиальных алгоритмов (в предположении, что $P \neq NP$). Следовательно, точные методы могут требовать экспоненциального времени вычислений в худшем случае, что зачастую неприемлемо для практического использования. Поэтому в последние 50 лет приближенным методам решения задач КО уделяется всё больше внимания. При использовании приближенных методов нет гарантии, что будет найдено оптимальное решение, однако может быть найдено достаточно хорошее решение за значительно меньшее время [4, с. 269].

1.2.2 Общая характеристика метаэвристических алгоритмов

Метаэвристики — это общие эвристики, позволяющие находить близкие к оптимальным решения различных задач оптимизации за приемлемое время.

Различные описания метаэвристик в литературе позволяют сформулировать некоторые фундаментальные свойства, характеризующие все метаэвристики:

- Метаэвристики — это стратегии, которые «управляют» процессом поиска решения.

- Цель метаэвристики состоит в эффективном исследовании пространства поиска для нахождения оптимальных или близких к оптимальному решений.

- Метаэвристические алгоритмы варьируются от простых процедур локального поиска до сложных процессов обучения.

- Метаэвристические алгоритмы являются приближенными и, как правило, недетерминированными.

- Метаэвристические алгоритмы могут включать различные механизмы избегания попаданий в ловушку в ограниченной области пространства поиска.

- Метаэвристики могут быть описаны на абстрактном уровне и не быть предназначенными для решения конкретных задач.

- Метаэвристики могут использовать специфические знания об области поиска в виде эвристик, которые контролируются стратегией верхнего уровня.

- Современные метаэвристики используют сохраненный в памяти опыт поиска решения для управления дальнейшим поиском [18].

Стратегии поиска различных метаэвристик в значительной степени зависят от концепции самой метаэвристики. Существует несколько основных концепций метаэвристик. Некоторые из них могут быть описаны как «продвинутые» алгоритмы локального поиска. Цель метаэвристик данного типа — избежать застревания на локальном минимуме и продолжить исследование пространства решений, чтобы найти другой, потенциально более хороший, минимум [4, с. 271]. Примерами таких алгоритмов являются метод табу-поиска, метод поиска окрестностей переменных (VNS — Variable Neighborhood Search), метод имитации отжига, процедура жадного рандомизированного адаптивного поиска (GRASP — Greedy Randomized Adaptive Search Procedure). Эти алгоритмы начинают свою работу с начального решения, а затем на каждом шаге поиска текущее решение заменяется другим, найденным в окрестности текущего решения.

Другая концепция метаэвристик представлена в популяционных алгоритмах. Такие методы на каждой итерации имеют дело не с одним решением, а сразу с множеством решений (т.е. с популяцией). Поскольку данные методы имеют дело с совокупностью решений, популяционные алгоритмы предоставляют естественный, присущий только им способ исследования пространства поиска. Тем не менее, эффективность конкретных методов сильно зависит от способа обработки популяции [4, с. 284]. Примерами популяционных алгоритмов являются генетические алгоритмы, эволюционные стратегии, генетическое программирование, метод оптимизации муравьиной колонии [18].

Для реализации в данной работе был выбран один из наиболее эффективных метаэвристических алгоритмов — метод имитации отжига, который описан подробнее ниже.

1.2.3 Метод имитации отжига

Основные принципы метода

Метод имитации отжига основан на параллелизме между задачей нахождения минимума функции нескольких переменных и феноменом отжига в статистической механике. Отжиг — это процесс, в ходе которого твердое вещество, переведенное в жидкую фазу путем нагревания, возвращается в твердую фазу путем медленного снижения температуры таким образом, чтобы все частицы могли расположиться в идеальном кристаллизованном состоянии. Такое кристаллизованное состояние представляет собой глобальный минимум определенной энергетической функции.

Метод имитации отжига основан на рассмотрении целевой функции задачи минимизации как эквивалентной функции энергии некоторого воображаемого процесса отжига. В таком случае управляющий параметр T, называемый «температурой», используется для управления случайностью процесса поиска.

Суть метода имитации отжига может быть описана следующим образом: при поиске глобального минимума функции поиск должен осуществляться движением вниз в большинстве случаев, но не обязательно всегда (здесь под движением вниз понимается замена текущего решения на новое только в том случае, если значение целевой функции для нового решения меньше, чем для текущего).

Структура метода

Общий алгоритм метода имитации отжига можно описать как итеративную процедуру, состоящую из двух вложенных циклов. Внутренний цикл моделирует достижение теплового равновесия при данной температуре, поэтому назовем его циклом теплового равновесия. Внешний цикл выполняет процесс охлаждения, в ходе которого температура снижается, пока не будет достигнут критерий остановки и поиск не будет остановлен; этот цикл назовем циклом охлаждения или циклом отжига.

1. **Цикл теплового равновесия**

Начиная с начального решения, каждая итерация внутреннего цикла вычисляет новое решение, которое может быть принято или не принято с определенной вероятностью. Здесь можно выделить 3 элемента:

а) *Схема возмущений.*

Она определяет способ обновления решения. Сначала вычисляется «возмущение», а затем оно добавляется к текущему решению $\overline{x}$, чтобы получить новое решение $\overline{x}'$. Хотя возмущение всегда вычисляется случайным образом, это можно сделать, используя различные виды распределений. Самый простой способ — использовать равномерное распределение на допустимом множестве в пространстве решений; однако основная проблема такого способа заключается в том, что он требует очень медленного процесса охлаждения.

б) *Критерий принятия.*

Он определяет, будет ли текущее решение заменено новым. Наиболее распространенным критерием принятия является алгоритм Metropolis. В соответствии с ним «изменение энергии» ΔE вычисляется как разность значений функции энергии в новом и текущем состояниях (решениях):

$$\Delta E = E(\overline{x}') - E(\overline{x}) \tag{1.5}$$

где $E(\overline{x})$ соответствует значению целевой функции для решения $\overline{x}$. В таком случае, если $\Delta E < 0$, новое решение всегда принимается; но если $\Delta E \geqslant 0$, то новое решение принимается с вероятностью, которая вычисляется по формуле:

$$P(\Delta E) = \exp\left(-\frac{\Delta E}{T}\right) \tag{1.6}$$

где T — это температура. С другой стороны, если новое решение не принимается, новая итерация начнется с тем же начальным решением $\overline{x}$.

в) *Достижение теплового равновесия.*

Как было сказано выше, на каждой итерации цикла теплового равновесия вычисляется новое решение, которое затем принимается или отклоняется в соответствии с критерием принятия, после чего начинается следующая итерация. Этот процесс повторяется снова и снова, пока не будет решено, что тепловое равновесие достигнуто.

2. **Цикл охлаждения**

Цикл охлаждения или цикл отжига — это внешний цикл алгоритма. Он начинается со случайно выбранного начального решения и начального значения температуры. На каждой итерации температура постепенно снижается, пока не будет выполнен определенный критерий сходимости. Опять же, здесь можно выделить 3 элемента:

а) *Начальная температура.*

Начальное значение параметра температуры имеет решающее значение для успешной работы алгоритма. Низкая начальная температура может привести к потере глобального характера поиска, поскольку поиск ограничится областью пространства решений вокруг начальной точки. С другой стороны, слишком высокая начальная температура приведет к тому, что алгоритм будет совершать «случайные блуждания» по пространству решений в течение большого числа итераций. Это приведет к ненужной трате вычислительного времени, и, что еще хуже, это может привести к неудачному поиску, если общее число итераций ограничено.

б) *График охлаждения.*

Он определяет способ снижения температуры. Это также играет важную роль для успешности поиска. При очень медленном охлаждении потребуется много итераций для достижения глобального минимума, и, если общее количество итераций ограничено, поиск может закончиться неудачей. С другой стороны, слишком быстрое охлаждение может привести к тому, что алгоритм застрянет в локальном минимуме.

Одним из распространенных процессов охлаждения является логарифмический график:

$$T_k = \frac{\alpha T_0}{\ln(1+k)} \tag{1.7}$$

где T_k — значение температуры на итерации k, T_0 — начальное значение температуры, а α — параметр скорости охлаждения. Для этого графика доказана сходимость к глобальному минимуму при $\alpha = 1$ [6]. Тем не менее, этот график охлаждения настолько медленный, что редко используется на практике. Хотя использование значений $\alpha < 1$ может ускорить процесс, логарифмические графики охлаждения, как правило, считаются слишком медленными.

Другим распространенным графиком охлаждения, более часто используе-

мым на практике, является геометрический график:

$$T_k = \alpha^k T_0 \tag{1.8}$$

В таком типе графика охлаждения параметр α должен быть меньше, но близким к 1. Самые типичные значения α — между 0,8 и 0,99; меньшие значения могут привести к слишком быстрому охлаждению.

Наконец, еще один распространенный график охлаждения — экспоненциальный:

$$T_k = T_0 \exp\left(-\alpha k^{\frac{1}{N}}\right) \tag{1.9}$$

где N — размерность пространства решений. Этот тип графика охлаждения является очень быстрым в течение первых итераций, но экспоненциальная скорость снижения температуры может быть уменьшена при использовании значений $\alpha < 1$.

в) *Критерий остановки.*

Одним из распространенных критериев остановки является достижение определенного числа принятий новых решений для некоторого числа последовательных значений температуры [3]. Также процесс поиска решения может быть завершен при достижении определенного количества итераций или потраченного времени.

1.3 Задача программирования в ограничениях

1.3.1 Определения и постановка задачи

Рассмотрим вначале определение задачи удовлетворения ограничений:

Определение 6. Задача удовлетворения ограничений — это упорядоченная пара вида $CSP = (C, D)$, где $D = D_1 \times ... \times D_n$ — область определения конечного числа переменных $x_j \in D_j, j = \overline{1,n}$; $C = \{C_1, ..., C_m\}$ — конечное множество ограничений $C_i : D \to \{0, 1\}, i \in \overline{1,m}$. Решить задачу удовлетворения ограничений — значит определить, является ли множество

$$X_{CSP} = \{x | x \in D, C(x)\}, \text{где } C(x) = \text{«истина»} \Leftrightarrow \forall i = \overline{1,m} : C_i(x) = 1 \tag{1.10}$$

непустым, т.е. либо найти решение $x \in D$, удовлетворяющее $C(x)$, либо доказать, что таких решений не существует [1, с. 9].

Оптимизационная версия задачи удовлетворения ограничений называется задачей программирования в ограничениях:

Определение 7. Задача программирования в ограничениях — это упорядоченная тройка вида $CP = (C, D, f)$, задача состоит в том, чтобы найти

$$f^* = \min \{f(x) | x \in D, C(x)\} \tag{1.11}$$

где $D = D_1 \times ... \times D_n$ — область определения переменных, $C = \{C_1, ..., C_m\}$ — множество ограничений, $f : D \to \mathbb{R}$ — целевая функция.

Множеством допустимых решений будем называть множество

$$X_{CP} = \{x | x \in D, C(x)\} \tag{1.12}$$

В общем случае задача программирования в ограничениях является NP-трудной. Показать это можно, например, таким образом: задача целочисленного линейного программирования является частным случаем задачи программирования в ограничениях, а для задачи ЦЛП, в свою очередь, доказано, что в общем случае она является NP-трудной [8, с. 232-235].

1.3.2 Общая схема решения задачи

Для решения задачи программирования в ограничениях обычно используются методы, зависящие от области определения переменных, либо общие методы, либо некоторая комбинация первых и вторых.

Методы, зависящие от области определения переменных, включают в себя реализации некоторых алгоритмов для конкретных областей определения переменных, например, следующие:

- программа, решающая системы линейных уравнений;
- реализация функций для линейного программирования;
- решатель целочисленного линейного программирования.

Общие методы, в свою очередь, направлены на уменьшение пространства поиска и на различные методы поиска. Такие методы работают для любых областей определения переменных и ограничений любого вида.

Одной из основных целей решателя задач программирования в ограничениях является поиск эффективных методов, зависящих от области определения переменных, которые могут быть использованы вместо общих методов, и встраивание их в общий ход решения. Этим целям уделяется большое внимание, поскольку методы, зависящие от области определения переменных, обычно являются гораздо более эффективными, нежели общие методы.

Опишем общую схему решения задачи удовлетворения ограничений в виде описанного в Алгоритме 1 псевдокода [2, с. 59, рисунок 3.1].

Алгоритм 1 Общий алгоритм решения задачи удовлетворения ограничений

Solve:

```
continue ← TRUE
while continue & not Happy do
    Preprocess
    Constraint Propagation
    if not Happy then
        if Atomic then
            continue ← FALSE
        else
            Split
            Proceed by Cases
        end if
    end if
end while
```

К сформулированной задаче удовлетворения ограничений последовательно применяется процедура $Solve$, описанная ниже. Она параметризована дополнительными процедурами $Preprocess$, $Constraint\ Propagation$, $Happy$, $Atomic$, $Split$, $Proceed\ by\ Cases$. Процедура $Proceed\ by\ Cases$ приводит к рекурсивному вызову процедуры $Solve$ для каждой сгенерированной в процессе решения подзадачи исходной задачи. Бинарная переменная $continue$ является локальной для процедуры $Solve$, следовательно, она заново определяется каждый раз, когда рекурсивно вызывается процедура $Solve$.

1.3.3 Этапы решения задачи

Опишем значение всех процедур, которые используются в процедуре $Solve$ в Алгоритме 1.

Процедура *Preprocess*

Цель этой процедуры — привести рассматриваемую задачу удовлетворения ограничений к определенной синтаксической форме. Например, для ограничений, в который участвуют только бинарные переменные, удобной формой может быть конъюнктивная нормальная форма.

Обычно процедура $Preprocess$ вызывается только один раз — при первом вызове процедуры $Solve$ (для исходной задачи). В Алгоритме 1 процедура $Preprocess$ помещена внутрь процедуры $Solve$, чтобы рассмотреть ситуацию, когда необходимо применить процедуру $Preprocess$ после вызова процедуры $Split$: процедура $Split$ может сгенерировать ограничения, которые нужно преобразовать, чтобы подзадача имела необходимый вид.

Процедура *Constraint Propagation*

В общих чертах, процедура $Constraint\ Propagation$ заменяет данную задачу удовлетворения ограничений на в каком-то смысле более простую, но эквивалентную исходной, задачу. Смысл заключается в том, что такая замена является полезной, так как последующий поиск решения осуществляется на меньшем пространстве поиска. Что означает «более простая» задача, зависит от конкретной задачи. Обычно это значит, что области определения переменных и / или некоторые ограничения становятся меньше. Процесс распространения ограничений заключается в повторном уменьшении областей определения переменных и / или уменьшении ограничений, при условии поддержания эквивалентности исходной задаче.

Идея процесса распространения ограничений заключается в том, что после применения процедуры $Split$ у некоторых переменных меняется область определения. Учитывая новые области определения, некоторые ограничения задачи могут оказаться, например, невыполнимыми или верными при любых значениях переменных. Тогда в первом случае можно сразу заключить, что данная подзадача не имеет решений, и на этом завершить ее рассмотрение; во втором

же случае можно убрать такое ограничение из постановки задачи, поскольку оно является излишним. Другим примером распространения ограничений может быть уточнение области значений какой-либо переменной: с учетом новых ограничений, полученных после применения процедуры $Split$, может оказаться, что при некоторых значениях какой-то переменной существуют ограничения, которые всегда нарушаются. Это говорит о том, что область определения такой переменной можно уменьшить, убрав из нее те значения, при которых задача не имеет решения.

Таким образом, распространение ограничений является центральной концепцией в теории и практике программирования в ограничениях. Важность распространения ограничений заключается в том, что эта процедура может значительно упростить задачу и, следовательно, повысить эффективность поиска решения [10, с. 4].

Процедура *Happy*

В общем случае процедура $Happy$ возвращает бинарное значение, которое означает, была ли достигнута цель исходной задачи. Наиболее распространенными целями задачи являются следующие:

- найдено допустимое решение;
- найдены все допустимые решения;
- доказано, что задача не имеет решений;
- найдено оптимальное по отношению к некоторой целевой функции решение;
- (в случае, если переменные являются вещественными числами) все области определения, являющиеся промежутками, имеют длину меньше некоторого заранее зафиксированного ϵ.

Процедура $Happy$ может рассматриваться как тест, применяемый к текущей подзадаче. В этом тесте могут также приниматься во внимание некоторые дополнительные параметры в том случае, если в задаче требуется найти оптимальное по отношению к некоторой целевой функции решение.

Процедура *Atomic*

Прежде чем разбивать задачу на более мелкие подзадачи, нужно проверить, что текущую задачу можно разбить. Эта проверка и происходит в процедуре $Atomic$. Обычно полагают, что текущую подзадачу нельзя разбить на более мелкие подзадачи, если область определения переменных этой подзадачи состоит из одного элемента или является пустым множеством. Однако задача может считаться неразбиваемой, если дальнейший поиск решения в этой подзадаче не нужен. Например, это может происходить в случае, когда текущая подзадача уже решена, или (при условии, что требуется найти оптимальное решение) оптимальное решение может быть вычислено напрямую.

Процедура *Split*

В этой процедуре текущая задача разбивается на 2 или более подзадачи, объединение которых дает исходную задачу. Новые подзадачи отличаются от текущей тем, что имеют меньшие области определения переменных.

Приведем несколько распространенных примеров того, как можно разбить область определения некоторой переменной в исходной задаче.

- Предположим, что область определения D является конечной и содержит как минимум 2 элемента; $a \in D$. Тогда D можно разбить на множества $\{a\}$ и $D \backslash \{a\}$.

- Если некоторая переменная имеет область определения $\{a_1, a_2, ..., a_k\}$, то ее можно разбить на k одноэлементных множеств: $\{a_1\}$, $\{a_2\}$, ..., $\{a_k\}$.

- Допустим, область определения переменной является непустым вещественным промежутком $[a, b]$. Тогда такую область определения можно разбить, например, на 2 промежутка: $\left[a, \frac{a+b}{2}\right]$ и $\left[\frac{a+b}{2}, b\right]$. Ясно, что такое же разбиение можно применить и к непустым целочисленным промежуткам.

Процедура *Proceed by Cases*

Процедура $Split$ генерирует 2 или более новых подзадачи. В общем случае, из-за многократного применения процедуры $Split$ генерируется некоторое дерево новых подзадач. Цель процедуры $Proceed\ by\ Cases$ — обойти это дерево в определенном порядке и, если это необходимо, во время обхода обновлять соответствующие переменные в зависимости от новой информации, полученной

в процессе обхода дерева (в случае, если требуется найти оптимальное решение). Два наиболее известных способа обхода — это обход в глубину и метод ветвей и границ. Особенностью обоих способов является то, что дерево подзадач заранее неизвестно, оно генерируется во время обхода.

Приведем краткое сравнение описанных в данной главе методов решения задач. В отличие от ЦЛП и программирования в ограничениях, метод имитации отжига может найти решение задачи довольно быстро, однако оптимальность найденного решения не гарантирована, как не гарантировано и то, что допустимое решение будет найдено. Преимущество ЦЛП перед программированием в ограничениях заключается в том, что поскольку ЦЛП является частным случаем программирования в ограничениях, то решатели ЦЛП могут применять дополнительные алгоритмы, например, симплекс-метод для решения релаксаций и методы отсечения, такие как метод отсечения Гомори. Однако в терминах программирования в ограничениях для задачи можно построить математическую модель с более понятной и логичной структурой, что может помочь решателю сделать полезные выводы о задаче или принимать верные решения при ветвлении [1, с. 13].

Таким образом, в данной главе были рассмотрены три метода, которые могут быть использованы для решения задачи о планировании рейсов воздушных судов: целочисленное линейное программирование, метод имитации отжига и программирование в ограничениях; каждый из этих методов имет свои достоинства и недостатки. В следующей главе описаны математические модели, построенные для решения задачи каждым из представленных выше методов.

ГЛАВА 2

РЕШЕНИЕ ЗАДАЧИ О ПЛАНИРОВАНИИ РЕЙСОВ ВОЗДУШНЫХ СУДОВ

2.1 Постановка задачи

Рассмотрим задачу планирования рейсов воздушных судов в следующей постановке.

Имеется расписание рейсов воздушных судов на какой-то относительно небольшой промежуток времени (на неделю) и доступные самолеты нескольких типов (самолеты одного типа считаются одинаковыми). Требуется каждому рейсу назначить конкретный самолет, минимизировав суммарную стоимость перелетов и стоянок в аэропортах.

Чтобы использовать полученное расписание на практике, достаточно добавить следующее условие: для каждого самолета аэропорт прибытия последнего рейса совпадает с аэропортом отправления первого рейса. При таком условии найденное расписание самолетов можно зациклить и использовать на протяжении довольно длительного промежутка времени.

Ниже приведены примеры параметров, которыми могут характеризоваться сущности данной задачи.

Среди параметров для рейсов могут рассматриваться:

- аэропорт отправления;
- аэропорт прибытия;
- время отправления;
- время прибытия;
- расстояние между аэропортами отправления и прибытия;
- минимальное число пассажиров, которое необходимо перевезти этим рейсом.

Среди параметров для типов самолетов можно выделить следующие:

- количество самолетов данного типа;
- количество мест в самолете данного типа;

- максимальное расстояние, которое может пролететь самолет данного типа;
- цена перелета единицы расстояния или расход топлива.

Параметрами аэропортов могут быть следующие:
- цена стоянки за единицу времени в данном аэропорту;
- цена топлива в данном аэропорту.

Помимо описанных выше параметров в действительности существует еще множество других. Кроме того, существует 4 типа проверок работоспособности, которые каждый самолет периодически должен проходить. Например, самая частая проверка проводится каждые 500 часов полета и занимает от 20 часов до 2 суток. Необходимость выделять время на проверки для каждого самолета существенно влияет на назначение рейсам самолетов. Однако в данной работе необходимость проверок учитываться не будет.

2.2 Подходы к решению задачи

В общем виде задача о планировании рейсов воздушных судов может включать в себя различные подзадачи: составление расписания рейсов, назначение рейсам конкретных самолетов, планирование работы экипажа, перепланировка графика полетов самолетов и экипажа в случае возникновения сбоев или изменений в исходном расписании. В современной литературе описано множество подходов для решения задачи о планировании рейсов воздушных судов в той или иной формулировке. Основными подходами являются целочисленное или смешанно-целочисленное программирование, динамическое программирование, метаэвристические методы (генетические алгоритмы, метод имитации отжига, муравьиный алгоритм и т.д.) и обучение с подкреплением [7].

При рассмотрении задачи в описанной в предыдущем разделе формулировке можно выделить следующие 2 подхода к решению.

1. На первом этапе назначить каждому рейсу тип самолета, минимизировав при этом суммарную стоимость перелетов. На втором этапе имеем для каждого типа самолетов свое расписание рейсов, для которого каждому рейсу нужно назначить конкретный самолет. Поскольку все самолеты одного типа являются одинаковыми, на втором этапе не будет функции, значение которой нужно минимизировать, поэтому решением задачи будет любое допустимое назначение самолетов рейсам [9, 13].

2. Все самолеты изначально считаются различными, каждый самолет имеет характеристики того типа самолетов, к которому он относится, и сразу решается исходная задача.

Первый подход, в отличие от второго, использует информацию о том, что есть одинаковые самолеты, следовательно, он потенциально более эффективный. Однако второй подход проще для реализации, а также, в отличие от первого подхода, при использовании второго подхода если у задачи существует решение, оно гарантированно будет найдено.

В данной работе задача решена в соответствии со вторым подходом.

2.3 Формализация входных данных

Заметим, что самолет не может прилететь и сразу вылететь на следующий рейс, ведь ему как минимум нужно сначала высадить пассажиров с предыдущего рейса, потом заправиться и принять пассажиров на следующий рейс. Значит, между прилетом и вылетом самолета должно пройти какое-то время. Поэтому ко всем исходным временам прибытия прибавим длину какого-то фиксированного временного промежутка и от всех времен отправления отнимем длину такого же временного промежутка (в дальнейшем будем считать, что эта операция уже проделана), тогда данное замечание будет учтено и при построении математической модели можно будет считать, что самолету разрешено прилететь и сразу же вылететь на следующий рейс. Пусть длина этого промежутка равна m единиц времени.

Формализуем входные данные:

- f_i — структура, описывающая i-й рейс. Она включает в себя следующие значения:

– $f_i.dep\_point$ — номер аэропорта отправления;

– $f_i.arr\_point$ — номер аэропорта прибытия;

– $f_i.dep\_time$ — время отправления;

– $f_i.arr\_time$ — время прибытия;

– $f_i.dist$ — расстояние перелета;

– $f_i.min\_pass$ — минимальное число пассажиров, которое необходимо перевезти этим рейсом.

Количество рейсов равно I.

- a_j — структура, описывающая j-й самолет. Она включает в себя следую-

щие значения:

– $a_j.seats$ — количество мест в самолете;

– $a_j.cost$ — цена перелета единицы расстояния.

Количество самолетов равно J.

- p_l — структура, описывающая l-й аэропорт. Она включает в себя следующие значения:

– $p_l.cost$ — цена стоянки за единицу времени.

Количество аэропортов равно L.

Отметим, что связанные характеристики $f_i.min\_pass$ и $a_j.seats$ можно рассматривать в более общем виде как некую величину, которая у самолета должна быть не меньше, чем у рейса, на который он назначен. Помимо числа пассажиров, такой величиной может быть, например, расстояние: максимальная дальность полета у самолета должна быть не меньше, чем расстояние перелета у рейса.

2.4 Решение задачи с помощью ЦЛП

2.4.1 Математическая модель задачи

Построим по исходным данным несколько массивов вспомогательных данных.

1. Составим массив из времен отправления по всем рейсам, отсортируем его по неубыванию и оставим лишь уникальные элементы. Аналогично получим массив из времен прибытия. Сольем полученные 2 массива так, чтобы полученный массив T был отсортирован по неубыванию. Добавим к массиву T в начало значение 0, в конец — значение, равное длине временного промежутка расписания из условия. Получим массив всех моментов времени, причем если в некоторый момент времени какой-то самолет прилетает и какой-то вылетает, то в массиве T этот момент времени встречается дважды. Пусть далее K — количество элементов в массиве T.

Для удобства в исходных данных заменим $f_i.dep\_time$ на индекс этого момента времени в массиве T, причем если в T такой момент времени встречается дважды, возьмем индекс, соответствующий второму вхождению. Аналогично заменим $f_i.arr\_time$ на индекс этого момента времени в массиве T, причем если в T такой момент времени встречается дважды, возьмем индекс, соответ-

ствующий первому вхождению.

2. Вычислим значения $F_{ik} \in \{0,1\}$, $i = \overline{1,I}, k = \overline{1,K-1}$, следующим образом: $F_{ik} = 1$, если в промежутке времени $[T_k, T_{k+1}]$ самолет, назначенный i-му рейсу, находится в воздухе; в противном случае $F_{ik} = 0$.

3. Вычислим значения $D_{il} \in \{0,1\}$, $i = \overline{1,I}, l = \overline{1,L}$, следующим образом: $D_{il} = 1$, если номер аэропорта отправления i-го рейса равен l; в противном случае $D_{il} = 0$.

4. Вычислим значения $R_{il} \in \{0,1\}$, $i = \overline{1,I}, l = \overline{1,L}$, следующим образом: $R_{il} = 1$, если номер аэропорта прибытия i-го рейса равен l; в противном случае $R_{il} = 0$.

Введем следующие переменные:

- $x_{ij} \in \{0,1\}$, $i = \overline{1,I}, j = \overline{1,J}$. Значение $x_{ij} = 1$ означает, что рейсу i назначен самолет j; $x_{ij} = 0$ означает противное.

- $A_{ljk} \in \{0,1\}$, $l = \overline{1,L}, j = \overline{1,J}, k \in \overline{1,K-1}$. Значение $A_{ljk} = 1$ означает, что самолет j в промежутке времени $[T_k, T_{k+1}]$ стоит в аэропорту с номером l; $A_{ljk} = 0$ означает противное.

Зададим множество планов задачи с помощью следующих ограничений (далее M — это какое-то наперед заданное большое число):

1. $\sum\limits_{j=1}^{J} x_{ij} = 1,\ i = \overline{1,I}$

Описание: каждому рейсу назначен ровно один самолет.

2. $x_{ij} \cdot a_j.seats \geqslant x_{ij} \cdot f_i.min\_pass,\ i = \overline{1,I}, j = \overline{1,J}$

Описание: если рейсу i назначен самолет j, то в самолете j мест не меньше, чем требуется перевезти рейсом i.

3. $|A_{ljk} - D_{il}| \leqslant M(1 - x_{ij}),\ k = f_i.dep\_time - 1, i = \overline{1,I}, j = \overline{1,J}, l = \overline{1,L}$

Описание: в промежутке времени непосредственно перед вылетом j-й самолет стоял в аэропорту отправления i-го рейса.

4. $|A_{ljk} - R_{il}| \leqslant M(1 - x_{ij}),\ k = f_i.arr\_time, i = \overline{1,I}, j = \overline{1,J}, l = \overline{1,L}$

Описание: в промежутке времени непосредственно после прилета j-й самолет стоял в аэропорту отправления i-го рейса.

5. $|A_{ljk} - A_{ljk+1}| \leqslant M \sum\limits_{i=1}^{I} x_{ij}(F_{ik} + F_{ik+1}),\ l = \overline{1,L}, j = \overline{1,J}, k = \overline{1,K-2}$

Описание: если в двух соседних промежутках времени j-й самолет не летел, то он стоял в одном и том же аэропорту.

6. $A_{lj0} = A_{ljK-1},\ l = \overline{1,L}, j = \overline{1,J}$

Описание: каждый самолет в конце расписания стоит в том же аэропорту, что и вначале.

7. $\sum\limits_{i=1}^{I} x_{ij} \cdot F_{ik} \leqslant 1,\ j = \overline{1, J}, k = \overline{1, K-1}$

Описание: каждому самолету назначено множество непересекающихся по времени рейсов.

8. $\sum\limits_{l=1}^{L} A_{ljk} = 1 - \sum\limits_{i=1}^{I} x_{ij} \cdot F_{ik},\ j = \overline{1, J}, k = \overline{1, K-1}$

Описание: если в промежутке времени $[T_k, T_{k+1}]$ j-й самолет летит, то в этом промежутке времени он не стоит ни в одном аэропорту; в противном случае j-й самолет стоит ровно в одном аэропорту.

Корректность 3 и 4 ограничений следует из построения массива T: если есть 2 рейса i_1, i_2 таких, что время отправления рейса i_1 совпадает с временем прибытия рейса i_2, то в T этот момент времени встречается дважды. Более того, в таком случае $f_{i_2}.dep\_time < f_{i_1}.arr\_time$ (как индексы в массиве T), и если какому-то самолету назначить оба этих рейса, то в промежутке времени $[f_{i_2}.dep\_time, f_{i_1}.arr\_time]$ (нулевой длины) этот самолет будет стоять на земле.

Определим целевую функцию, которую будем минимизировать. Она включает следующие слагаемые:

1. $\sum\limits_{i=1}^{I} \sum\limits_{j=1}^{J} x_{ij} \cdot a_j.cost \cdot f_i.dist$

Описание: суммарная стоимость перелетов.

2. $\sum\limits_{j=1}^{J} \sum\limits_{k=1}^{K-1} \sum\limits_{l=1}^{L} A_{ljk} \cdot p_l.cost \cdot (T_{k+1} - T_k + 2m)$

Описание: суммарная стоимость стоянок во всех аэропортах. $+2m$ в скобках отвечает за те 2 промежутка времени, которые в начале построения модели были включены во время полета (m единиц времени перед вылетом и m — после), ведь на самом деле эти $2m$ едениц времени самолет стоит в каком-то аэропорту, поэтому они тоже учитываются в суммарной стоимости.

Из области определения переменных, ограничений и целевой функции видно, что построенная модель является задачей ЦЛП, поэтому в дальнейшем эта задача будет решаться соответствующими методами.

2.4.2 Оценка полученной модели как задачи ЦЛП

Оценим полученную модель, а именно оценим количество переменных, ограничений и ненулевых коэффициентов.

- Количество переменных равно $IJ + LJ(K-1)$.

- Вычислим суммарное количество ограничений (количество строк в матрице ограничений задачи ЦЛП). Первое ограничение модели дает I строк, второе — IJ, третье и четвертое — по $2ILJ$ (после раскрытия модуля из каждого неравенства получается 2), пятое — $LJ(K-2)$, шестое — LJ, седьмое и восьмое — по $J(K-1)$. Итого получаем $I + IJ + LJK - 2LJ + LJ + 2JK - 2J =$ $= I + J(I + 4IL + LK - L + 2K - 2) = O(JL(I+K))$ строк.

- Вычислим количество ненулевых коэффициентов для каждого ограничения модели:

1. В каждой строке J коэффициентов, равных 1, всего строк I, следовательно, это ограничение дает IJ коэффициентов.

2. В каждой строке по 2 ненулевых коэффициента, строк IJ, значит в этом ограничении $2IJ$ коэффициентов.

3. Из-за модуля каждое неравенство дает 2 строки, в каждой строке 2 ненулевых коэффициента, откуда общее число коэффициентов равно $4IJL$.

4. Аналогично предыдущему пункту $4IJL$ коэффициентов.

5. В каждой строке левая часть дает 2 коэффициента, при всех $k = \overline{1, K-2}$ и фиксированных l, k правая часть дает не больше, чем $2\sum\limits_{i=1}^{I}\sum\limits_{k=1}^{K-1} F_{ik} \leqslant 2HI$ ненулевых коэффициента, где H — максимальное по всем рейсам количество промежутков времени, в течение которых самолет, назначенный рейсу, находится в воздухе. При раскрытии модуля число строк удваивается, следовательно, всего получаем не более $2(2LJ(K-2)+2HILJ) \leqslant 4LJ(K+HI)$ ненулевых коэффициентов.

6. LJ строк, в каждой из которых по 2 коэффициента, значит, всего $2LJ$ коэффициента.

7. При фиксированном j и всех $k = \overline{1, K-1}$ получаем $\sum\limits_{i=1}^{I}\sum\limits_{k=1}^{K-1} F_{ik} \leqslant HI$ ненулевых коэффициентов, следовательно, всего их не более HIJ.

8. Аналогично предыдущему пункту правая часть дает не более HIJ ненулевых коэффициентов, левая часть дает $LJ(K-1)$ коэффициентов, значит, всего их не более $J(HI + LK)$.

Суммарно получаем не более $IJ + 2IJ + 4IJL + 4IJL + 4LJK + 4HIJL +$ $+ 2LJ + HIJ + HIJ + LJK = O(JL(IH + K)$ ненулевых коэффициентов.

В целом, главный показатель размера задачи ЦЛП — количество ненулевых

коэффициентов — получился довольно большим: IJL с довольно большой константой и LJK даже на не слишком больших данных (например, $I \approx 400$, $J \approx 30$, $L \approx 30$, $K \approx 150$) могут представить проблему для современных решателей.

2.5 Решение задачи с помощью метода имитации отжига

2.5.1 Математическая модель задачи

Построим по исходным данным вспомогательный массив T следующим образом. Составим массив из времен отправления и прибытия по всем рейсам, отсортируем его по неубыванию и оставим лишь уникальные элементы, получим массив всех временных точек, встречающихся в расписании. Пусть далее K — количество элементов в массиве T.

Для удобства в исходных данных заменим $f_i.dep\_time$ и $f_i.arr\_time$ на индексы этих моментов времени в массиве T.

Введем следующие переменные:

- $y_i \in \{1, 2, ..., J\}$, $i = \overline{1, I}$. y_i — номер самолета, назначенного на i-й рейс.

Введем функцию штрафа, которую нужно будет минимизировать. Опишем слагаемые, которые в нее входят, и структуры данных, с помощью которых эти слагаемые будут вычисляться.

1. *Описание:* если на i-й рейс назначен самолет y_i, то в этом самолете мест не меньше, чем $f_i.min\_pass$.

Формула: $\max\{f_i.min\_pass - a_{y_i}.seats, 0\}$. Тогда при корректном назначении самолетов на рейсы это слагаемое будет равно нулю.

2. *Описание:* каждому самолету назначено множество непересекающихся по времени рейсов.

Введем таблицу B, состоящую из J строк и $K - 1$ столбцов. Значение B_{jk} будет означать количество рейсов, которые j-й самолет пролетает в k-й промежуток времени. Дополнительно будем хранить счетчик элементов таблицы, значение в которых больше 1. Тогда корректному назначению самолетов на рейсы будет соответствовать нулевое значение этого счетчика. Данный счетчик с некоторым коэффициентом и будет слагаемым в функции штрафа.

При изменении одной компоненты y_i вектора решения y нужно уменьшить на 1 значения всех элементов таблицы, соответствующих прежнему значению y_i, а затем добавить 1 к элементам таблицы, соответствующим новому

значению y_i. При необходимости корректируется значение счетчика.

3. *Описание:* для каждого самолета аэропорт прибытия предыдущего рейса совпадает с аэропортом отправления следующего рейса. А также аэропорт вылета первого рейса совпадает с аэропортом прибытия последнего рейса, что дает условие цикличности расписания.

Введем массив размера J из сбалансированных бинарных деревьев поиска. Тогда j-е дерево будет содержать временные точки отправлений и прибытий всех рейсов, назначенных j-му самолету. Ключ в каждом дереве — множество из временной точки, стадии (отправление или прибытие) и номера аэропорта; сравнение ключей идет сначала по времени, при равенстве — по стадии (прибытие должно быть раньше отправления), затем при равенстве по номеру аэропорта. Значение для каждого ключа — штраф за несовпадение аэропортов (0 или 1). Условимся, что для k-го ключа штраф равен 1 только в следующем случае: стадия на k-м ключе — прибытие, на $k+1$-м — отправление и аэропорты на этих ключах не совпадают. Для соблюдения условия зацикливания будем считать, что следующим ключом после последнего является первый. Будем также дополнительно хранить суммарный штраф по всем деревьям — он и будет слагаемым в штрафной функции.

Для добавления ключа в дерево сначала нужно найти место, в которое будет вставлен ключ, обнулить штраф на ключе перед вставляемым, вставить новый ключ и при необходимости поставить штраф 1 на предыдущий ключ либо на вставленный.

Для удаления ключа из дерева нужно обнулить штрафы на удаляемым ключе и на ключе, предшествующем ему, удалить ключ и при необходимости поставить штраф 1 на предыдущий ключ.

4. *Описание:* суммарная стоимость перелетов.

Формула: $\sum\limits_{i=1}^{I} a_{y_i}.cost \cdot f_i.dist$.

5. *Описание:* суммарная стоимость стоянок во всех аэропортах.

Будем использовать те же деревья, что и в пункте 3. В каждом дереве каждый k-й ключ будет давать следующий штраф:

$$\begin{cases} (T_{k+1} - T_k + 2m) \cdot p_l.cost, & \text{если стадия на } k\text{-м ключе — прибытие, а на } k+1\text{-м — отправление} \\ 0, & \text{иначе} \end{cases}$$

Аналогично предыдущей модели, $+2m$ в скобках отвечает за те 2 временных промежутка, которые в начале построения модели были включены во время полета (m единиц времени перед вылетом и m — после).

Заметим, что если слагаемое в пункте 2 не равно 0 (это равносильно тому, что есть самолет, на который назначены пересекающиеся по времени рейсы), то в пунктах 3 и 5 невозможно корректно определить штраф, поэтому эти слагаемые не будут отражать реальные значения штрафа.

Решим эту проблему следующим образом: умножим слагаемое из пункта 2 на какой-то коэффициент M_1, слагаемое из пункта 3 — на коэффициент M_2, из пункта 1 — на M_3, из пунктов 4 и 5 — на M_4, и пусть $M_1 >> M_2 >> M_3 >> M_4$. Тогда если в пункте 2 штраф не равен 0, то суммарно получаем очень большой штраф вне зависимости от того, что посчитано в остальных пунктах. Если же штраф в пункте 2 равен 0, то суммарный штраф становится значительно меньше, а в пунктах 3 и 5 уже посчитано корректное значение. Значение M_3 выбрано так, что $M_3 << M_2$, поскольку ограничение из пункта 3 является гораздо более существенным, чем ограничение из пункта 1. Наконец, $M_4 << M_3$, поскольку минимизация (а именно равенство 0) слагаемых из пунктов 1–3 дает какое-то, возможно не оптимальное, но допустимое решение, а слагаемые из пунктов 4–5 позволяют найти оптимальное решение, но никак не влияют на его допустимость. То есть при минимизации штрафа следует вначале занулить 1–3 слагаемые, а уже потом пытаться минимизировать 4–5.

2.5.2 Оценка эффективности модели

Оценим полученную модель, а именно:

- затраты по количеству дополнительной памяти;
- затраты по времени для перевычисления каждого слагаемого 1–5 в штрафной функции при изменении одной компоненты вектора решения.

Второй пункт интересен именно в таком виде, поскольку в данной работе задача решается методом имитации отжига, в ходе которого нужно уметь быстро пересчитывать штраф при изменении одной компоненты вектора решения. Вычислим затраты по времени и по памяти для каждого слагаемого.

1. Здесь достаточно хранить текущее значение штрафа, тогда новое значение пересчитывается за время $O(1)$. Дополнительной памяти тремуется $O(1)$.

2. Количество времени, которое требуется для пересчитывания штрафа,

пропорционально числу временных точек, содержащихся между точками отправления и прибытия рейса, которому назначается другой самолет. Если положить, что рейсы распределены по расписанию более-менее равномерно, то можно считать, что это число ограничено константой, т.е. штраф пересчитывается за время $O(1)$. В худшем же случае это занимает время $O(K)$.

Для вычислений используется таблица размера $J \times K$, поэтому дополнительной памяти требуется $O(JK)$.

3. Максимальное число элементов в одном дереве равно $2I$ (если все рейсы назначены одному самолету). Значит, удаление и вставка элемента занимают $O(\log I)$ времени, а поскольку для изменения одной компоненты вектора решения требуется удалить 2 элемента и вставить 2 элемента, суммарно пересчет функции штрафа занимает $O(\log I)$ времени.

Каждый рейс порождает 2 элемента в каком-то дереве, поэтому общее число элементов в деревьях равно $2I$. Но поскольку всего деревьев J, общее число дополнительной памяти равно $O(I + J)$.

4. Здесь, как и в пункте 1, достаточно хранить текущее значение штрафа, и тогда новое значение пересчитывается за время $O(1)$. Дополнительной памяти тремуется $O(1)$.

5. Для вычисления этого слагаемого используются те же деревья, что и для слагаемого из пункта 3, штраф пересчитывается аналогичным пункту 3 образом, поэтому пересчет также занимает $O(\log I)$ времени.

Дополнительной памяти (помимо той, что уже посчитана в пункте 3) требуется $O(1)$ — для хранения текущего значения штрафа.

Таким образом, суммарные затраты следующие:

- *Время пересчета штрафа:* $O(\log I)$ (если рейсы распределены по расписанию равномерно) или $O(K + \log I)$ — в худшем случае.

- *Дополнительная память:* $O(I + JK)$.

В целом, если считать, что рейсы распределены по расписанию более-менее равномерно, то штраф пересчитывается довольно быстро. Это позволит при использовании метода имитации отжига проделывать довольно большое число итераций за разумное время, что повышает вероятность нахождения как минимум допустимого решения.

2.6 Решение задачи с помощью программирования в ограничениях

При составлении моделей задачи в терминах программирования в ограничениях будем считать, что рейсы отсортированы по неубыванию времен отправления.

2.6.1 Первый вариант математической модели задачи

Введем следующие переменные:

- $S_j \subseteq \{1, 2, ..., I\}$, $j = \overline{1, J}$. S_j — это множество номеров рейсов, которые назначены j-му самолету. Для удобства далее будем считать, что все S_j отсортированы по неубыванию.

Введем следующие ограничения на переменные:

1. $\bigsqcup_{j=1}^{J} S_j = \{1, 2, ..., I\}$

Описание: каждый рейс назначен ровно одному самолету.

2. $\forall j = \overline{1, J}, \forall i \in S_j$:

$f_i.min\_pass \leqslant a_j.seats$

Описание: если рейсу i назначен самолет j, то в самолете j мест не меньше, чем требуется перевезти рейсом i.

3. $\forall j = \overline{1, J}, \forall k = \overline{1, |S_j| - 1} : i_1 = S_j[k], i_2 = S_j[k + 1]$:

$f_{i_1}.arr\_time \leqslant f_{i_2}.dep\_time$

Описание: каждому самолету назначено множество не пересекающихся по времени рейсов. Поскольку рейсы отсортированы по неубыванию времени отправления, среди рейсов, назначенных одному самолету, неравенство рассматривается лишь для соседних по индексам (а значит, и по времени отправления) рейсов.

4. $\forall j = \overline{1, J}, \forall k = \overline{1, |S_j| - 1} : i_1 = S_j[k], i_2 = S_j[k + 1]$:

$f_{i_1}.arr\_point = f_{i_2}.dep\_point$

Описание: для каждого самолета аэропорт прибытия предыдущего рейса совпадает с аэропортом отправления следующего рейса.

5. $\forall j = \overline{1, J}, i_1 = S_j[1], i_2 = S_j[|S_j|]$:

$f_{i_2}.arr\_point = f_{i_1}.dep\_point$

Описание: каждый самолет последним назначенным ему рейсом прилетает в тот же аэропорт, из которого вылетает первым рейсом.

Введем целевую функцию, которую будем минимизировать. Она складывается из следующих слагаемых:

1. $\sum\limits_{j=1}^{J} \sum\limits_{i \in S_j} f_i.dist \cdot a_j.cost$

Описание: суммарная стоимость всех перелетов.

2. $\sum\limits_{j=1}^{J} \sum\limits_{k=1}^{|S_j|-1} \big(f_{S_j[k+1]}.dep\_time - f_{S_j[k]}.arr\_time + 2m\big) \cdot p[f_{S_j[k]}.arr\_point].cost$

Описание: суммарная стоимость стоянок в аэропортах в промежутках между прибытием первого для самолета рейса и отправлением последнего для самолета рейса.

3. $\sum\limits_{j=1}^{J} \big(finish\_time - f_{S_j[|S_j|]}.arr\_time + f_{S_j[1]}.dep\_time - start\_time + 2m\big) \times$
$\times p[f_{S_j[1]}.dep\_point].cost$

Описание: суммарная стоимость стоянок в аэропортах в промежутках между прибытием последнего для самолета рейса и отправлением первого для самолета рейса. Здесь введены дополнительные константы: $start\_time$ — время начала расписания, $finish\_time$ — время окончания расписания.

Аналогично предыдущим моделям, в последних двух слагаемых целевой функции $+2m$ в скобках отвечает за те 2 промежутка времени, которые в начале построения модели были включены во время полета (m единиц времени перед вылетом и m — после).

2.6.2 Второй вариант математической модели задачи

Введем следующие переменные:

- $c_i \in \{1, 2, ..., J\},\ i = \overline{1, I}$. c_i — номер самолета, назначенного на i-й рейс.

А также введем несколько вспомогательных переменных:

- $n_i \in \{0, 1, ..., J\},\ i = \overline{1, I}$. n_i — номер ближайшего следующего рейса, на который назначен тот же самолет, что и на рейс i, или 0, если такого рейса нет.

- $first_j \in \{0, 1, ..., I\},\ j = \overline{1, J}$. $first_j$ — номер первого рейса, который назначен j-му самолету, или 0, если j-му самолету не назначен ни один рейс.

- $last_j \in \{0, 1, ..., I\},\ j = \overline{1, J}$. $last_j$ — номер последнего рейса, который назначен j-му самолету, или 0, если j-му самолету не назначен ни один рейс.

Введем следующие ограничения на переменные (первые 3 ограничения задают определения вспомогательных переменных):

1. $\forall i = \overline{1, I}$:

$$n_i = \begin{cases} \min\{i' : i' > i, c_i = c_{i'}\}, \text{если такое } i' \text{ существует;} \\ 0, \text{в противном случае.} \end{cases}$$

2. $\forall j = \overline{1, J}$:

$$first_j = \begin{cases} \min\{i : c_i = j\}, \text{если такое } i \text{ существует;} \\ 0, \text{если самолету } j \text{ не назначен ни один рейс.} \end{cases}$$

3. $\forall j = \overline{1, J}$:

$$last_j = \begin{cases} \max\{i : c_i = j\}, \text{если такое } i \text{ существует;} \\ 0, \text{если самолету } j \text{ не назначен ни один рейс.} \end{cases}$$

4. $\forall i = \overline{1, I}$:

$f_i.min\_pass \leqslant a_{c_i}.seats$

Описание: если рейсу i назначен самолет j, то в самолете j мест не меньше, чем требуется перевезти рейсом i.

5. $\forall i = \overline{1, I}$:

$f_i.arr\_time \leqslant f_{n_i}.dep\_time$, если $n_i \neq 0$

Описание: соседние по времени рейсы, назначенные одному и тому же самолету, не пересекаются по времени.

6. $\forall i = \overline{1, I}$:

$f_i.arr\_point \leqslant f_{n_i}.dep\_point$, если $n_i \neq 0$

Описание: для каждого самолета и для каждых двух последовательных рейсов, которые ему назначены, аэропорт прибытия первого такого рейса совпадает с аэропортом отправления второго рейса.

7. $\forall j = \overline{1, J}$:

$f_{last_j}.arr\_point \leqslant f_{first_j}.dep\_point$, если $first_j \neq 0$

Описание: каждый самолет последним назначенным ему рейсом прилетает в тот же аэропорт, из которого вылетает первым рейсом.

Введем целевую функцию, которую будем минимизировать. Она складывается из следующих слагаемых:

1. $\sum\limits_{i=1}^{I} f_i.dist \cdot a_{c_i}.cost$

Описание: суммарная стоимость всех перелетов.

2. $\sum\limits_{j=1}^{J} \sum\limits_{i=1}^{last_j - 1} I\{c_i = j\} \cdot (f_{n_i}.dep\_time - f_i.arr\_time + 2m) \cdot p[f_i.arr\_point].cost$

Здесь $I\{A\}$ — функция, которая вводится следующим образом:

$$I\{A\} = \begin{cases} 1, \text{если выражение } A \text{ истинно;} \\ 0, \text{если выражение } A \text{ ложно.} \end{cases}$$

Описание: суммарная стоимость стоянок в аэропортах в промежутках между прибытием первого для самолета рейса и отправлением последнего для самолета рейса.

3. $\sum\limits_{j=1}^{J} \big(finish\_time - f_{last_j}.arr\_time + f_{first_j}.dep\_time - start\_time + 2m\big) \times$

$\times\, p[f_{first_j}.dep\_point].cost$

Описание: суммарная стоимость стоянок в аэропортах в промежутках между прибытием последнего для самолета рейса и отправлением первого для самолета рейса.

2.6.3 Оценка эффективности моделей

В общем случае, предпочтительнее строить модель таким образом, чтобы в ней присутствовало как можно меньше переменных и чтобы ограничения были по возможности более простыми и легко проверяемыми, чтобы для них хорошо работал алгоритм распространения изменений [2, с. 357]. Кроме того, чем меньше область определения переменных, тем меньше существует различных вариантов значений набора переменных, что может существенно влиять на размер дерева поиска.

Отметим, что, как показывает практика, в поставленной задаче существует очень мало допустимых решений. Из этого следует, что сложность вычисления целевой функции практически не играет роли, поскольку целевая функция вычисляется лишь для допустимых решений с целью найти среди них оптимальное.

Теперь отдельно оценим эффективность обеих построенных моделей.

1. Оценим для начала области определения переменных. Каждая переменная S_j имеет область определения $2^{\{1,2,...,I\}}$, т.е. включает в себя 2^I значений. Таким образом, для набора переменных $S_j, j = \overline{1, J}$ возможно $(2^I)^J = 2^{IJ}$ значений. Однако первое ограничение указывает на то, что переменные S_j задают разбиение множества $\{1, 2, ..., I\}$ на J подмножеств, и решатели могут распознавать данное ограничение и делать такой вывод, поэтому по сути получаем J^I возможных значений для набора переменных S_j.

Заметим, что количество переменных в предложенной модели невелико.

Однако некоторые ограничения (например, 4 и 5) могут представлять проблему для решателей, поскольку представление переменной типа «множество» может не предусматривать упорядоченность элементов. В таком случае, проверка ограничений, которые опираются на порядок элементов в S_j, может занимать значительное время. К тому же, в таком случае данные переменные могут быть неэффективными для алгоритма распространения изменений. В остальном, однако, все ограничения (кроме первого) — это ограничения типа равенства или неравенства, т.е. ограничения являются довольно простыми.

Заметим, что количество ограничений в данной модели варьируется в зависимости от значений переменных S_j. Однако оценим приблизительно общее количество ограничений для каждой группы ограничений, описанных выше.

1) В этом пункте присутствует 1 ограничение.

2) Заметим, что в этой группе ограничений каждый рейс встречается ровно 1 раз, следовательно, в этой группе I ограничений.

3) Для каждого множества $S_j \neq \emptyset$ имеем $|S_j| - 1$ ограничений. Таким образом, в худшем случае получим $I - 1$ ограничение (в ситуации, когда все множества S_j, кроме одного, являются пустыми).

4) Аналогично предыдущему пункту, здесь имеем в худшем случае $I - 1$ ограничение.

5) Для каждого j имеем 1 ограничение, следовательно, всего имеем J ограничений.

Таким образом, в худшем случае всего имеем $1 + I + (I - 1) + (I - 1) + J = O(I + J)$ ограничений. Число ограничений линейно зависит от I и от J, причем с маленькой константой, следовательно, при фиксированных значениях переменных S_j проверка удовлетворения ограничений займет относительно немного времени.

2. Оценим области определения переменных. c_i может принимать J значений, следовательно, набор c_i может принимать J^I значений. Заметим, что по значениям c_i значения вспомогательных переменных определяются однозначно, вычисление этих значений можно даже перенести в ограничения, тогда из переменных останутся только c_i. К тому же при поиске решения имеет смысл производить ветвления только по переменным c_i, и тогда остальные переменные не будут влиять на размер дерева поиска. Поэтому в дальнейшем для оценки не будем учитывать области определения вспомогательных переменных. Та-

ким образом, количество переменных невелико, сами переменные из множества подряд идущих целых чисел, что хорошо подходит для ветвления.

Ограничения 4 – 7 представляют из себя неравенства, т.е. являются достаточно простыми ограничениями. Однако ограничения, которые определяют значения вспомогательных переменных, являются довольно сложными, поскольку для их проверки требуется либо последовательно просматривать массив c_i, либо для каждого самолета составлять множества рейсов, которые он пролетает. Такие ограничения являются для решателей довольно нетривиальными.

Найдем общее количество ограничений. Группы ограничений с номерами 1, 4, 5, 6 включают в себя по I ограничений, остальные группы — по J ограничений. Всего получаем $4I + 3J = O(I + J)$ ограничений. Число ограничений линейно зависит от I и от J, причем с маленькой константой, значит, при фиксированных значениях переменных проверка удовлетворения ограничений займет относительно немного времени.

Подытоживая вышесказанное, сделаем следующие выводы. У обеих моделей число переменных и ограничений достаточно невелико, что должно хорошо отразиться на времени решения. Однако области определения переменных по размеру довольно большие, и это может представлять проблему для решателей, поскольку чем больше область определения переменных, тем больше будет дерево поиска решения. К тому же при увеличении размера задачи множество значений в области определения переменных возрастает экспоненциально, следовательно, для больших размеров входных данных (например, при $I = 50, J = 5$ получаем уже $J^I = 5^{50}$ возможных вариантов для набора переменных S_j или c_i) построенные модели на практике для больших задач скорее всего будет неприменимы. Кроме того, в обеих моделях присутствуют довольно сложные ограничения, что также может отрицательно сказаться на времени решения задачи.

Таким образом, каждая из построенных в данной главе математических моделей задачи имеет свои достоинства и недостатки, и исходя только из теоретических оценок нельзя понять, насколько эффективно каждое решение — для этого требуется провести вычислительный эксперимент. В следующей главе описаны алгоритмы генерации тестовых данных, реализация построенных моделей, а также их тестирование, анализ и сравнение.

ГЛАВА 3

ПРАКТИЧЕСКИЕ РЕЗУЛЬТАТЫ

3.1 Генерация тестовых данных

Для генерации одного тестового расписания следующие параметры нужно задать вручную:

- количество рейсов в расписании;
- количество типов самолетов;
- количество доступных самолетов;
- количество аэропортов;
- длительность расписания;
- m — длина промежутка времени, который самолет должен стоять в аэропорту перед вылетом и после прилета;
- минимальное и максимальное возможные расстояния между аэропортами;
- средняя скорость самолета (для вычисления времени полета);
- минимальное и максимальное числа пассажиров, которых нужно перевезти данными рейсами; на основе этих чисел генерируются также и количества мест в самолетах;
- минимальная и максимальная цены за перелет единицы расстояния;
- минимальная и максимальная цены стоянки в аэропортах за единицу времени;
- минимальное и максимальное количества рейсов на один самолет (используется во втором алгоритме). Данные параметры позволяют гарантировать, что построенное во втором алгоритме назначение самолетов на рейсы не нарушит санитарные нормы: количество летных часов для самолета за определенный промежуток времени не будет превышать указанную норму.

Все расстояния и промежутки времени должны быть указаны в одних и тех же единицах измерения.

3.1.1 Первый вариант алгоритма

Если отождествить аэропорты с вершинами графа, а рейсы — с дугами, то исходное расписание будет представлять собой ориентированный мультиграф без петель с дополнительной информацией на дугах. Поэтому вначале сгенерируем ориентированный мультиграф с количеством вершин, равным количеству аэропортов, и количеством дуг, равным количеству рейсов. Естественно предположить, что в действительности если есть рейс из пункта A в пункт B, то есть и рейс из пункта B в пункт A. Поэтому сгенерируем мультиграф так, чтобы количества противоположно направленных дуг между каждыми двумя вершинами графа совпадали (отсюда возникает естественное требование: количество рейсов должно быть четным числом).

Вначале случайным образом генерируется массив D степеней выхода вершин таким образом, что сумма степеней выхода (равная количеству дуг в искомом графе) равна количеству рейсов, причем если количество рейсов не меньше количества аэропортов, то все степени выхода выбираются не меньше 1. Далее генерируется случайный ориентированный мультиграф с заданными для вершин степенями выхода в соответствии с описанным ниже рекурсивным алгоритмом, построенным на основе алгоритма из статьи [16].

1. Заведем матрицу смежности G искомого графа, в которой элемент G_{ij} равен количеству дуг (i, j) в текущем графе. Изначально матрица G заполнена нулями.

2. Построим матрицу A, в которой элемент A_{ij} равен количеству дуг, которые еще можно провести из вершины i в вершину j. Изначально A_{ij} равно максимуму из степеней выхода данных вершин.

3. Если R — массив степеней выхода вершин графа G — совпадает с D с точностью до перестановки элементов, то граф построен, задача решена. Если после сортировки D и $R\ \exists j : R_j > D_j$, то в этой ветви рекурсии граф не найден, решение завершено.

4. На каждом шаге выбираем вершину i с наименьшей степенью, после чего выбираем вершину j с вероятностью $P = \frac{d_j}{N}$, где d_j равно $D_j - G_{ij}$, если $A_{ij} > 0$, иначе $d_j = 0$, $N = \sum\limits_j d_j$.

5. Решаем 2 независимые подзадачи:

а) Уменьшаем на 1 значения A_{ij} и A_{ji}, решаем исходную задачу.

б) Увеличиваем G_{ij} и G_{ji} на 1, решаем исходную задачу.

Таким образом, сгенерированы пункты отправления и прибытия рейсов. Далее генерируются времена отправления и прибытия. Минимально возможное время отправления равно m, максимально возможное время прибытия — количество единиц времени в расписании минус m. Так в расписании учитывается, что перед вылетом и после прилета самолет должен стоять в аэропорту хотя бы m единиц времени. Времена отправления генерируются случайным образом, времена прибытия рассчитываются исходя из времен отправления, расстояния между аэропортами и средней скорости самолета. Расстояния между аэропортами генерируются таким образом, чтобы для расстояний между любыми тремя аэропортами выполнялось неравенство треугольника. Остальные данные генерируются случайным образом исходя из заданных вручную ограничений. Все случайные величины берутся из дискретного равномерного распределения.

На практике оказалось, что данный алгоритм генерирует тестовые расписания так, что примерно в 95% расписаний не существует допустимых решений. Поэтому был разработан другой алгоритм, описанный ниже.

3.1.2 Второй вариант алгоритма

Данный алгоритм построен таким образом, чтобы гарантиировать, что у каждого сгенерированного расписания будет хотя бы одно допустимое решение. Идея алгоритма основывается на том, чтобы сгенерировать расписание для каждого самолета отдельно, а потом объединить эти расписания в одно. Опишем этот алгоритм подробнее.

1. По заданному числу аэропортов генерируется матрица попарных расстояний между всеми аэропортами, причем таким образом, что для расстояний между любыми тремя аэропортами выполняется неравенство треугольника.

2. Для каждого самолета генерируется количество рейсов, которые он будет пролетать, причем $(2n+1)$-й и $(2n+2)$-й самолеты ($\forall n \in \{0, 1, ..., \lfloor \frac{J}{2} \rfloor - 1\}$) пролетают одинаковое число рейсов.

3. Для каждого $(2n+1)$-го самолета генерируется последовательность аэропортов, где каждые 2 соседних аэропорта различны; количество эаропортов в последовательности равно количеству рейсов, который пролетает данный самолет. Далее для каждого самолета генерируются рейсы: i-й рейс — это рейс из i-го аэропорта последовательности в $(i+1)$-й аэропорт (последний рейс — из последнего аэропорта последовательности в первый). Времена отправления

и прибытия для рейсов генерируются так, чтобы рейсы шли в заданной выше последовательности и не пересекались по времени.

4. Для каждого $(2n + 2)$-го самолета последовательность рейсов строится следующим образом: берется последовательность рейсов предыдущего самолета, у каждого рейса аэропорты отправления и прибытия меняются местами, и порядок следования рейсов меняется на противоположный. После этого последовательность рейсов остается правильной (т.е. аэропорт прибытия предыдущего рейса совпадает с аэропортом отправления следующего рейса). Выполним циклический сдвиг последовательности рейсов на случайное число для внесения большей случайности в итоговое расписание. Потом перегенерируем времена отправления и прибытия в соответствии с полученной последовательностью. Количества пассажиров, которых нужно перевезти, остаются без изменений.

5. Если число самолетов нечетное, то для последнего самолета последовательность аэропортов генерируется следующим образом (пусть число рейсов для этого самолета равно p; заметим, что оно четное, поскольку общее число рейсов четно): первые $\frac{p}{2} + 1$ аэропортов последовательности генерируются случайным образом (с условием, что каждые 2 соседних аэропорта различны), оставшиеся $\frac{p}{2} - 1$ аэропортов — это аэропорты с 2-го по $(\frac{p}{2})$-го, записанные в обратном порядке. Далее аналогично пункту 3 генерируются рейсы, их времена отправления и прибытия. Для первых $\frac{p}{2}$ рейсов количества пассажиров генерируются случайным образом, для оставшихся рейсов число пассажиров для рейса i равно числу пассажиров для рейса $p - i + 1$.

6. Далее генерируются характеристики самолетов. Для каждого самолета известны рейсы, которые он пролетает, так что число мест в самолете — это случайное число, не меньшее всех количеств пассажиров на рейсах, которые пролетает данный самолет. Цена перелета единицы расстояния — случайное число.

7. Для каждого аэропорта случайным образом генерируется стоимость стоянки в этом аэропорту за единицу времени.

В описанном выше алгоритме все случайные числа берутся из равномерного распределения с соответствующими верхней и нижней границами.

Таким образом, вначале генерируется допустимое решение, а потом на основании этого решения строятся тестовые данные. Заметим, что для построен-

ного допустимого решения выполняются естественные условия, заданные для тестовых данных, а именно: для каждого рейса из аэропорта i в аэропорт j существует рейс из аэропорта j в аэропорт i с таким же количеством пассажиров, которых требуется перевезти.

3.1.3 Сгенерированные данные

Длительность всех тестовых расписаний — 7 дней, поскольку на практике большинство авиакомпаний составляют расписание рейсов на неделю. Количество аэропортов L бралось равным количеству самолетов J, количество рейсов бралось равным $I = 12J$, поскольку по проанализированным данным в среднем у различных авиакомпаний отношение числа рейсов к числу самолетов приблизительно равняется 12, а число самолетов примерно равняется числу аэропортов. Далее под размером теста будем понимать число самолетов J.

Для каждого размера были сгенерированы 2 набора тестов: набор тестов, имеющих решение, и набор тестов, не имеющих решения; в каждом наборе по 20 расписаний. Размеры тестов были выбраны следующие: 3, 5, 7, 10, 12, 15, 17, 20, 25, 28.

3.1.4 Получение реального расписания

Для анализа того, могут ли реальные авиакомпании использовать разработанные решения, требовалось получить расписание какой-нибудь авиакомпании. Для примера была выбрана авиакомпания Belavia.

Расписание авиакомпании Belavia взято из источника [14]. Данные html-страницы из этого источника были обработаны на языке Python с помощью библиотеки BeautifulSoup. Все времена отправления и прибытия были переведены в один часовой пояс и представлены в виде числа минут, прошедших с начала цикла расписания. После описанной обработки данные были приведены к тому виду, в котором их принимают описанные выше решения, и записаны в файл в формате json.

3.2 Реализация построенных решений

3.2.1 Реализация решения с помощью ЦЛП

В качестве решателя задачи ЦЛП был выбран один из самых мощных на данный момент открытых решателей — HiGHS. Предобработка данных и по-

строение модели реализованы на языке C++. Для работы с тестовыми данными, записанными в формате JSON, используется библиотека Poco. Для ускорения поиска решения HiGHS запускался с параметром *threads*, равным 10, что позволяет решателю работать многопоточно на 10 потоках.

3.2.2 Реализация решения с помощью метода имитации отжига

Реализация метода имитации отжига

Реализация данного решения написана на языке C++, для работы с тестовыми данными используется библиотека Poco. Для метода имитации отжига написаны 2 реализации:

1. с возможностью не назначать рейсу никакого самолета (за каждый рейс без самолета к суммарному штрафу добавляется настраиваемая константа);
2. без такой возможности.

Достоинство первого варианта в том, что он позволяет менять местами самолеты с настраиваемым штрафом, а также заданием небольшого штрафа за неназначенный самолет можно получить достижимое решение, требующее некоторое количество дополнительных самолетов. Преимущество же второго метода в том, что на любой итерации каждому рейсу будет назначен самолет.

Для метода имитации отжига реализованы 3 функции температуры (1.7), (1.8), (1.9), представленные в пункте 1.2.3. Для каждой функции нужно задать параметры T_0 и α.

Поскольку метод является рандомизированными, решение реализованы таким образом, чтобы несколько раз запускать решения с разными параметрами для генерации псевдослучайных чисел и выбирать среди них решение с минимальным штрафом. Это позволяет избежать ситуаций, когда найдено плохое решение только из-за того, что код запущен с неудачными параметрами для генерации случайных чисел. Чтобы время работы программы не увеличивалось кратно числу перезапусков, решение запускается параллельно на числе потоков, равном числу перезапусков. При тестирования это число равнялось 10.

Подбор наилучших параметров запуска

Для каждого размера тестовых данных подбиралась комбинация параметров α и T_0 среди следующих значений: $\alpha \in \{0,2; 0,4; 0,6; 0,8; 0,85; 0,9; 0,95;$

$0,97;1\}$ и $T_0 \in \{10, 100, 500, 1000, 1500, 2000\}$. Были найдены наилучшие параметры для всех трех функций температур (наилучшими считались параметры, для которых суммарный по всем тестам данного размера штраф был минимальным), число итераций при тестировании равнялось 10^6. Эти параметры приведены в таблице 1.

Таблица 1 — Наилучшие параметры для разных функций температуры

размер тестовых данных	не разрешать отсутствие самолета						разрешать отсутствие самолета					
	log		exp		geom		log		exp		geom	
	To	α	To	α	To	α	To	α	To	α	To	α
3	1000	0,4	1000	0,4	1000	0,85	500	0,9	500	0,9	500	0,97
5	10	0,9	10	0,9	10	0,97	500	0,4	500	0,4	500	0,85
7	500	0,6	500	0,6	500	0,9	500	0,2	500	0,2	500	0,8
10	1500	0,4	1500	0,4	1500	0,85	100	1	100	1	100	0,99
12	100	0,2	100	0,2	100	0,8	1000	0,9	1000	0,9	1000	0,97
15	1500	0,8	1500	0,8	1500	0,95	10	0,4	10	0,4	10	0,85
17	1500	0,6	2000	0,6	1500	0,9	1000	0,6	1000	0,6	1500	0,9
20	2000	0,4	2000	0,4	2000	0,85	2000	0,9	2000	1	2000	0,97
25	500	0,4	1500	0,4	500	0,85	1000	0,8	500	0,9	1000	0,95
28	2000	1	10	0,4	2000	0,99	1000	1	100	0,6	1000	0,99

3.2.3 Реализация решения с помощью программирования в ограничениях

Различные реализации моделей задачи

Учитывая специфику MiniZinc — языка описания моделей в терминах программирования в ограничениях — первую из описанных ранее моделей можно реализовать несколькими способами. Мотивация попробовать разные реализации модели заключается в том, что можно по-разному задать переменные и ограничения, и это приведет к деревьям поиска разного рзмера, что может в ту или иную сторону повлиять на скорость решения задачи [2, с. 356]. В документации MiniZinc рекомендуется по возможности использовать встроенные функции и предикаты, поскольку они зачастую реализованы оптимальнее, чем те же функции, записанные в терминах базовых функций [11]. Рассмотрим следующие варианты реализации.

1. Единственная переменная — массив элементов типа set размера J.

Достоинства. Для таких переменных легко задать некоторые ограничения. Например, для ограничения 1 существует встроенный предикат $partition\_set$, благодаря чему данное ограничение не нужно описывать с помощью нескольких отдельных ограничений в MiniZinc.

Недостатки. Некоторые ограничения модели в терминах MiniZinc записать довольно сложно. Например, ограничения 4, 5 требуют доступ к элементам

set-а по индексу, чего MiniZinc не предоставляет, из-за чего такая функция была реализована через другие ограничения. Кроме того, MiniZinc не гарантирует, что у переменной типа set задан какой-то порядок элементов, поэтому реализованный самостоятельно доступ к элементам set-а по индексу может заметно влиять на скорость поиска решения.

2. Единственная переменная — массив элементов типа set размера J. Эта реализация является попыткой оптимизировать запись ограничений 4, 5 из предыдущей реализации. Здесь ограничения 4, 5 для каждого set-а объединены в одно ограничение, в котором создается вспомогательный массив из элементов set-а, на него накладывается встроенное ограничение, что он отсортирован по неубыванию, после чего доступ к элементам осуществляется по индексу.

Достоинства. При проверке ограничений для каждого set-а элементы по всем индексам вычисляются один раз, после чего неоднократный доступ к элементам по индексу осуществляется быстро.

Недостатки. По сравнению с предыдущей реализацией, 2 ограничения слились в одно, внутри которого вводится локальная переменная типа массив. Сложность этого ограничения может негативно влиять на возможность распространения изменений и, следовательно, на отсечения по недопустимости некоторых ветвей в дереве поиска. Кроме того, в MiniZinc размер всех массивов должен быть известен на этапе компиляции, то есть в описанном выше ограничении нельзя задать массив того же размера, что и set. Из-за этого нужно задавать размер массива максимальным возможным, что равняется I — числу рейсов, и заполнять оставшиеся значения каким-нибудь значением, не равным ни одному номеру рейса (например, 0). Большой размер массива может негативно влиять на скорость сортировки.

3. Переменные: двумерный массив размера $J \times I$ (где каждая строчка — это номера рейсов, назначенные данному самолету, остальные элементы равны 0), 2 массива размера J (в одном указаны индексы первых ненулевых элементов в строках двумерного массива, во втором — количества ненулевых символов в строках двумерного массива). Эта реализация опирается на предыдущую, с той разницей, что здесь локальные переменные из ограничения предыдущей реализации являются глобальными.

Достоинства. В ограничениях не создаются локальные переменные, в ограничениях модели 4, 5 доступ по индексу осуществляется быстро (поскольку это

доступ по индексу в массиве). Кроме того, ограничение 1 задается двумя встроенными ограничениями на двумерный массив: *all_different_except_0* (которое проверяет, что все ненулевые элементы массива различны) и $count$ (проверяет, что количество нулей равно $I(J-1)$, что равносильно тому, что в массиве ровно I ненулевых элементов).

Недостатки. Двумерный массив занимает $O(IJ)$ памяти, причем $I(J-1)$ из элементов массива равны 0. Кроме того, каждая строка этого массива должна быть отсортирована, то есть на каждой проверке ограничений нужно отсортировать J массивов размера I, что может быть слишком долго.

Сравнение моделей

При тестировании моделей в терминах программирования в ограничениях в качестве решателя задачи программирования в ограничениях использовался Gecode 6.3.0, поскольку он показывал наилучшие по времени решения результаты.

Вначале было проведено сравнение реализаций первой модели между собой. Как показала практика, уже на самых маленьких тестах ($J = 3$) вторая реализация ищет решение примерно в 10 раз дольше, чем первая и третья. Разница между первой и третьей реализациями стала заметна при увеличении тестов: уже при входных данных $J = 7$ третья реализация работает дольше в среднем в 50 раз. Поэтому для дальнейших сравнений была выбрана первая реализация.

Далее было проведено сравнение первой и второй моделей между собой. Уже на тестах размера $J = 5$ решение, основанное на второй модели, работало в среднем в 10 раз быстрее, чем решение, основанное на первой модели. При увеличении размера тестов этот разрыв только увеличивался.

Следующим шагом был подбор наилучших опций запуска решателя. В документации MiniZinc [11] были выбраны следующие опции, которые могли ускорить поиск решения:

- *-p <n>* — запустить решение, используя *n* параллельных потоков.

- Аннотация *seq_search*. Эта аннотация позволяет указать массив других аннотаций, указывающих решателю, по каким переменным и с учетом каких правил нужно проводить ветвление. Учитывая, что во второй модели есть вспомогательные переменные, значения которых вычисляются через основные, одной из протестированных опций запуска было ветвление в первую очередь по

основным переменным.

- Различные варианты аннотации *restart*. Эта аннотация задает, на какой глубине дерева поиска следует начать поиск сначала (с течением времени эта глубина может меняться). Такой механизм позволяет не перебирать целиком потенциально неперспективные области дерева поиска.

По итогам тестирования второй модели с разными аннотациями или опциями запуска и без них получены следующие результаты:

- Аннотация *restart* не ускоряет поиск решения по сравнению с запуском без аннотаций. Уже на тестах размера $J = 7$ заметно, что реализация с использованием этой аннотации работает значительно медленнее (в 10 раз и больше), чем реализация без аннотаций.

- Реализация с аннотацией *seq_search* с указанием ветвления в первую очередь по основным переменным в среднем находит решение за то же время, что и реализация без аннотаций. Однако на тестах, где решения нет, в половине случаев реализация с аннотацией показывала гораздо большее время решения (от 3 до 100 раз) по сравнению с реализацией без аннотаций. Это можно объяснить тем, что без ограничения на выбор переменной для ветвления решатель может выбрать переменную более удачно, за счет чего будет проведено более удачное распространение изменений и отсечена ветвь по недопустимости.

- Запуск с использованием нескольких потоков всегда дает гораздо лучшие результаты, чем запуск на одном потоке.

Таким образом, для дальнейшего сравнения различных подходов к решению задачи планирования рейсов воздушных судов в качестве решения в терминах программирования в ограничениях будет использоваться описанная в пункте 2.6.2 модель. Поиск решения с использованием этой модели будет проводиться без использования каких-либо аннотаций, но с опцией запуска, позволяющей решателю использовать несколько параллельных потоков.

3.3 Тестирование решений и анализ результатов

Все тесты проводились на виртуальной машине со следующими характеристиками: операционная система Linux, процессор Intel Ice Lake (14 ядер), объем оперативной памяти — 28 ГБ.

3.3.1 Тестирование решения с помощью ЦЛП

При тестировании решения с помощью ЦЛП для каждого размера тестовых данных и для каждого набора данных этого размера (расписания, имеющие решение, и расписания, не имеющие решений) были найдены минимальное, максимальное и среднее значения времени поиска решения. Эти результаты представлены на графиках ниже (рисунок 3.1 и рисунок 3.2).

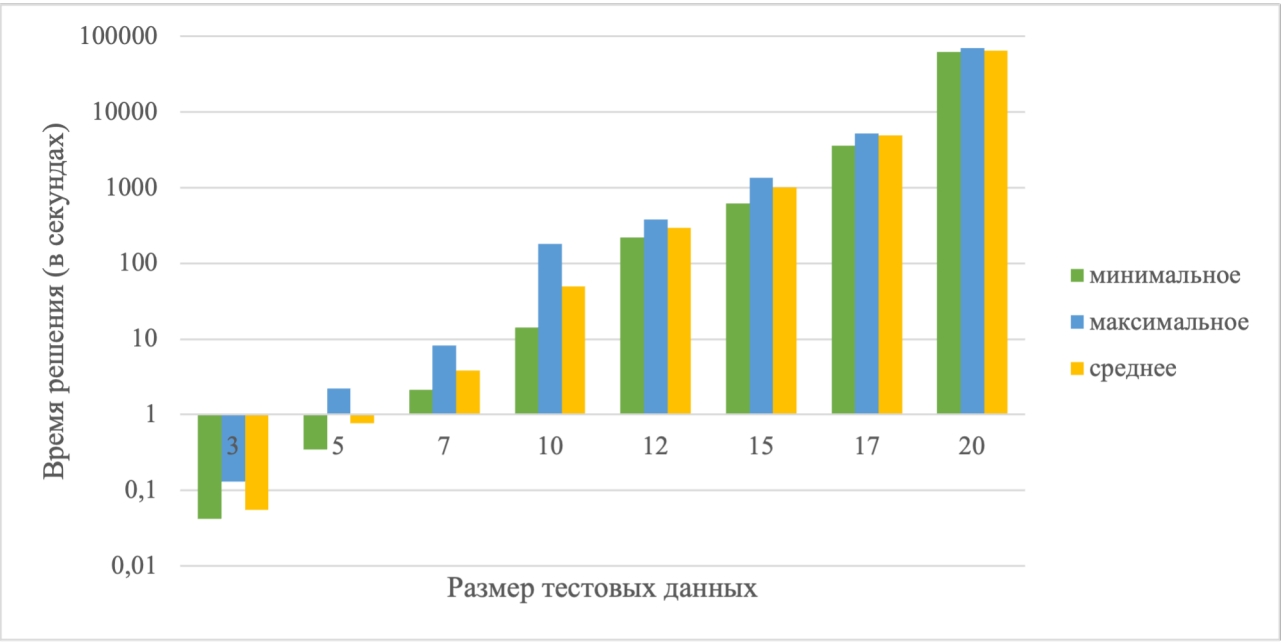

Рисунок 3.1 — Времена поиска решений (решения существуют)

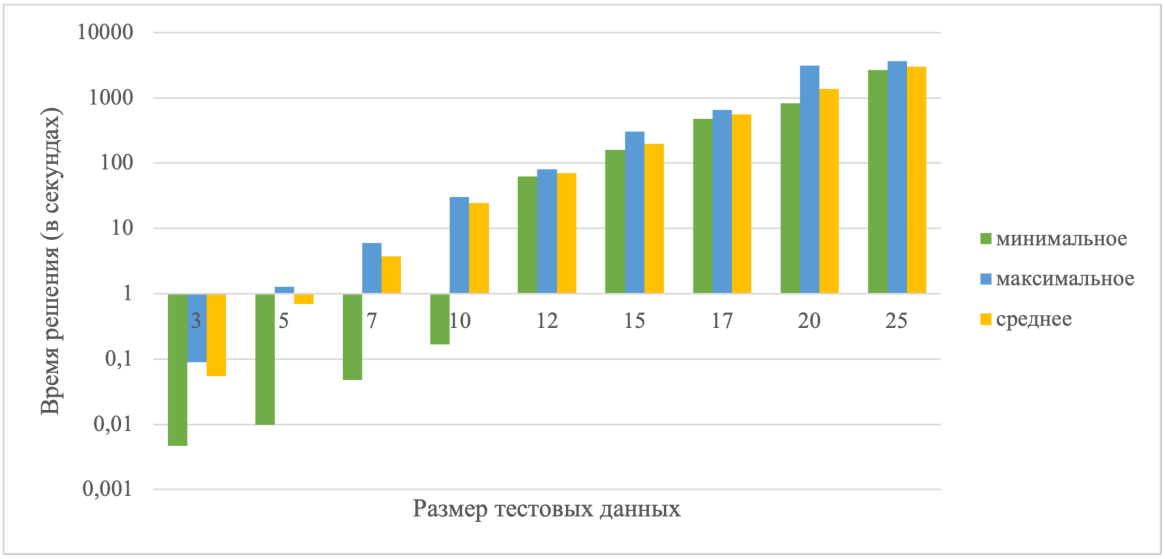

Рисунок 3.2 — Времена поиска решений (решений не существует)

Из представленных графиков видно, что при увеличении размера тестовых данных на 2-3 время решения растет экспоненциально (и в случае, когда решение существует, и в случае, когда решений нет). Ниже также приведен график сравнения средних времен решения для случая, когда решения существуют, и для случая, когда решений нет (рисунок 3.3).

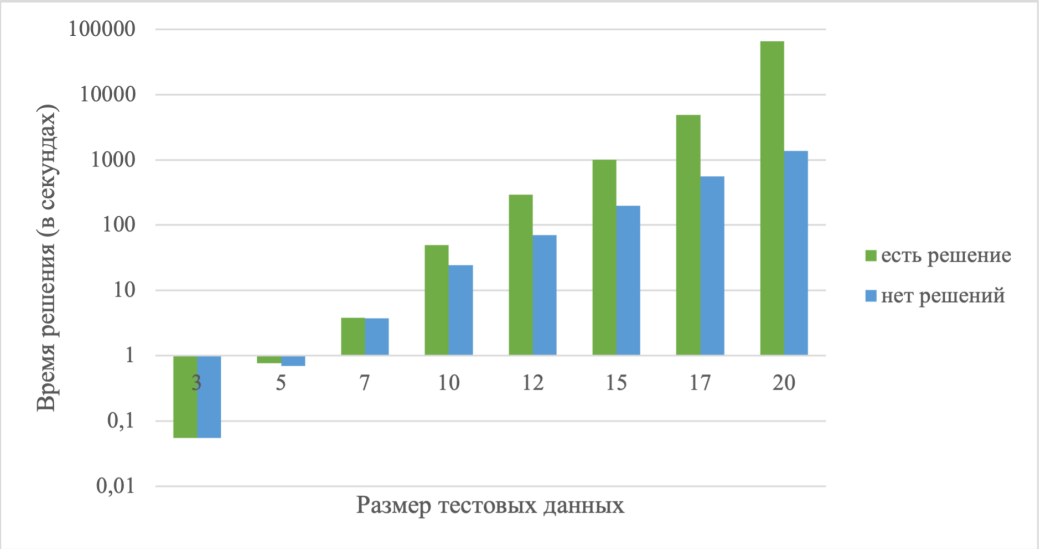

Рисунок 3.3 — Средние времена поиска решений: сравнение для тестов, где решения есть, и тестов, где решений нет

Из последнего графика (рисунок 3.3) видно, что когда решение существует, его поиск всегда работает дольше. Это можно объяснить тем, что если решения нет, то решатель может заметить это еще на небольшой глубине дерева поиска, например, в процессе распространения изменений. Если же решение существует, то в дереве поиска решателю придется пройти до листа, чтобы найти это решение. Кроме того, поскольку ищется оптимальное решение, требуется перебрать и оставшееся дерево поиска, возможно, еще где-то спустившись до листа.

3.3.2 Тестирование решения с помощью метода имитации отжига

Для каждого размера тестовых данных решение запускалось с наилучшими параметрами, подобранными в пункте 3.2.2. Были проверены различные числа итераций: от 10^3 до 10^8. Однако допустимые решения были найдены лишь в 20% тестовых расписаний (среди тестов, в которых решение существует), причем чем больше размер теста, тем в меньшем числе тестов было найдено решение.

При числе итераций 10^7 даже на больших тестах метод работает не более нескольких минут. Однако поскольку решение данный метод находит довольно редко, особенно с ростом размера входных данных, для решения исходной задачи данный метод применять нецелесообразно. Поэтому подробный анализ времени решения приводить не будем.

3.3.3 Тестирование решения с помощью программирования в ограничениях

При тестировании решения с помощью программирования в ограничениях были проанализированы результаты, аналогичные результатам тестирования с помощью ЦЛП (пункт 3.3.1). Эти результаты представлены на графиках ниже (рисунок 3.4 и рисунок 3.5). Отметим, что поскольку поиск оптимального решения работал значительно дольше, чем поиск допустимого решения, уже на тестах размера 5, на графике приведены времена работы для поиска допустимого решения.

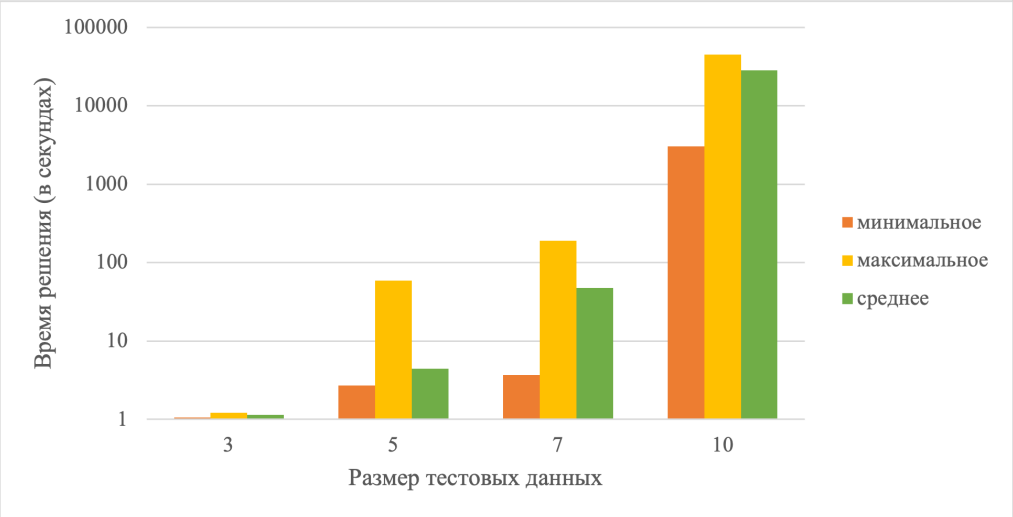

Рисунок 3.4 — Времена поиска решений (решения существуют)

Конечно, для более точного анализа на графиках представлено слишком мало данных, однако из них можно сделать предположение, что время решения с помощью программирования в ограничениях растет быстрее, чем экспоненциально. Ниже также приведен график сравнения средних времен решения для случая, когда решения существуют, и для случая, когда решений нет (рисунок 3.6).

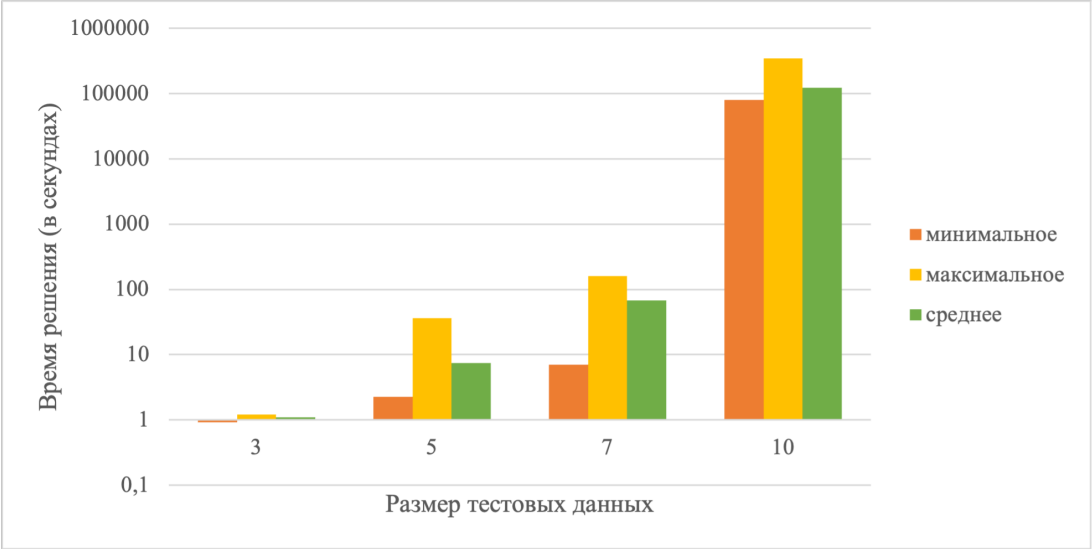

Рисунок 3.5 — Времена поиска решений (решений не существует)

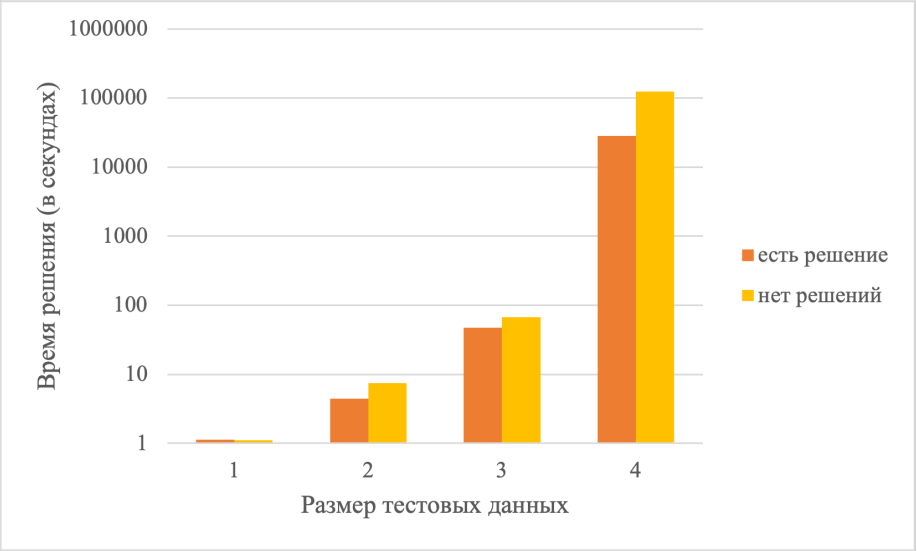

Рисунок 3.6 — Средние времена поиска решений: сравнение для тестов, где решения есть, и тестов, где решений нет

Из последнего графика (рисунок 3.6) видно, что в среднем решение работает быстрее, когда решение есть, чем когда решений нет. Это можно объяснить тем, что ищется допустимое решение, значит, если решение существует, то программа завершится сразу после того, как оно будет найдено, не обходя оставшееся дерево поиска.

3.3.4 Тестирование на реальном расписании

На полученном в пункте 3.1.4 расписании авиакомпании Belavia были протестированы решение с помощью ЦЛП и решение с помощью метода имитации отжига.

- С помощью метода имитации отжига не было найдено допустимое решение, несмотря на многократные перезапуски с разными функциями температуры и параметрами для них. Значение штрафной функции сходилось к одному и тому же значению, которое говорит о том, что в найденных решениях есть несовпадения аэропорта прибытия предыдущего рейса с аэропортом отправления следующего рейса для нескольких самолетов.
- С помощью ЦЛП допустимое решение было найдено за 3 дня, после чего решатель доказывал его оптимальность еще 4 дня. При небольших изменениях в расписании (уменьшении числа самолетов) решателю требовалось от 6 до 14 часов, чтобы установить, что решений не существует. В целом, время получения допустимого решения можно считать приемлемым.

Заметим, что если достаточно найти лишь допустимое решение или решение, которое хуже оптимального не более чем на какое-то значение, то можно указать решателю подходящее значение параметра gap, чтобы он не тратил время на доказательство оптимальности. Кроме того, имея допустимое решение задачи, мы можем его улучшить: ведь, имея допустимое решение, мы имеем разбиение множества рейсов на подмножества, пролетаемые одним самолетом. Если ограничения на число пассажиров довольно жесткие, то, учитывая, что число самолетов небольшое (в случае Belavia — 31), вычислив для каждого подмножества рейсов минимальную вместимость самолета, который на эти рейсы можно назначить, можно перебрать допустимые перестановки самолетов и вычислить для каждой значение целевой функции. Далее, выбирая перестановку с минимальным значением целевой функции, можно получить решение, более близкое к оптимальному.

3.3.5 Сравнение решений и анализ результатов

Из приведенных выше графиков видно, что наилучшим из описанных решений является решение с помощью ЦЛП, поскольку решение с помощью программирования в ограничениях всегда работает значительно дольше (а для размеров тестов, начиная с 12, и вовсе неприемлемо долго), а с помощью метода

имитации отжига — маловероятно, что какое-нибудь решение будет найдено.

Отметим, что в ходе тестирования решения с помощью ЦЛП обнаружилось, что хотя решателю разрешалось использовать 10 потоков, 99% времени реально он использовал не более 2 потоков. Кроме того, на всех тестах, в том числе на реальном расписании, решатель использовал не более 20 ГБ оперативной памяти. Эти наблюдения говорят о том, что в случае ЦЛП решение не удастся ускорить, увеличив вычислительные ресурсы компьютера (такие как число ядер процессора и количество оперативной памяти), на котором запускается решение. Значит, для дальнейшей оптимизации следует либо менять модель, либо добавлять в решатель некоторые специфичные для данной задачи алгоритмы.

Таким образом, построенное решение с помощью ЦЛП может успешно применяться авиакомпаниями с флотом до 20–25 самолетов. Авиакомпании с флотом до 30 самолетов также могут применять эту модель, однако для получения результата им потребуется несколько дней. То есть модель может использоваться малыми (с флотом до 20 самолетов) и некоторыми средними (с флотом до 30 самолетов) авиакомпаниями, которые составляют по разным данным от 65% до 90% всех авиакомпаний в мире.

ЗАКЛЮЧЕНИЕ

В данной дипломной работе была сформулирована задача назначения воздушных судов рейсам заранее известного расписания, описаны подходы к ее решению. Были построены 3 математические модели данной задачи: в терминах целочисленного линейного программирования, в терминах задачи комбинаторной оптимизации и в терминах задачи программирования в ограничениях.

Помимо этого были описаны 2 алгоритма генерации начальных расписаний, которые использовались для тестирования описанных моделей. Были сгенерированы тестовые начальные расписания различных размеров, найдена наилучшая по скорости и качеству решения модель задачи и подобран ориентировочный размер входных данных, начиная с которого эта модель решает задачу слишком долго.

В целом, два из трех описанных в данной работе решений задачи успешно решают поставленную задачу. Однако по результатам тестирования наиболее эффективным по времени работы решением является решение в терминах задачи ЦЛП. Оно успешно отрабатывает на небольших размерах данных за приемлемое время, поэтому на практике может использоваться малыми (с флотом до 20 самолетов) и некоторыми средними (с флотом до 30 самолетов) авиакомпаниями.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Achterberg, T. Constraint Integer Programming : Dissertation ... Doktor der Naturwissenschaften / T. Achterberg. – Berlin : Technischen Universität Berlin, 2007. – 412 p.
2. Apt, K. Principles of constraint programming / K. Apt – Amsterdam : Cambridge university press, 2003. – 407 p.
3. Banchs R. E. Simulated annealing / Banchs R. E. – Austin : University of Texas at Austin, 1997.
4. Blum C. Metaheuristics in combinatorial optimization: Overview and conceptual comparison / Blum C., Roli A. – New York : Association for Computing Machinery, 2003. – p.268-308.
5. Garey M. R. Computers and Intractability: A Guide to the Theory of NP-Completeness / M. R. Garey – San Francisco : Freeman, 1990. – p. 37-79.
6. Geman S. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images / Geman S., Geman D. – Piscataway : Institute of Electrical and Electronics Engineers, 1984. – p. 721-741.
7. Ikli, S. The aircraft runway scheduling problem: A survey / Ikli S. et al. // Computers & Operations Research. - Elsevier, 2021. - Vol. 132. - P. 105336.
8. Karp R. M. Reducibility among combinatorial problems / R. M. Karp. – Berlin : Springer Berlin Heidelberg, 2010. – p. 219-241.
9. Leung J. Y. T. Handbook of scheduling: algorithms, models, and performance analysis / Leung J. Y. T. (ed.) – Boca Raton : CRC press, 2004.
10. Rossi, F. Handbook of constraint programming / F. Rossi, P. Van Beek, T. Walsh (ed.) – Amsterdam : Elsevier, 2006. – 956 p.
11. The MiniZinc Handbook [Electronic resource] // MiniZinc information portal. – Mode of access: https://docs.minizinc.dev/en/stable/ – Date of access: 07.04.2025.

12. Wolsey L. A. Integer programming / L. A. Wolsey. – New York : John Wiley & Sons, 2021. – 2nd ed. – 316 p.

13. Zhou L. Airline planning and scheduling: Models and solution methodologies / Zhou, L., Liang, Z., Chou, CA. – Beijing : Frontiers of Engineering Management, 2020. – 26 p.

14. Информационный портал aviatickets [Электронный ресурс] // Режим доступа: https://aviatickets.by/airlines/belavia-belarusian-airlines-bru/ – Дата доступа: 03.03.2025.

15. Ковалёв, М. М. Дискретная оптимизация: Целочисленное программирование / М. М. Ковалёв. – М. : URSS, 2020. – 192 с.

16. Мельников Б. Ф. Генерация графов с заданным вектором степеней второго порядка и задача проверки изоморфизма / Мельников Б. Ф., Сайфуллина Е. Ф. – Санкт-Петербург : Издательство Санкт-Петербургского университета, 2014.

17. Методы оптимизации : пособие / Р. Габасов [и др.]. – Минск : Четыре четверти, 2011. – 472 с.

18. Щербина О. А. Метаэвристические алгоритмы для задач комбинаторной оптимизации (обзор) / Щербина О. А. – Симферополь : издательство Крымского федерального университета им. В. И. Вернадского, 2014. – с. 56-72.