

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS

Разработка системы автоматического планирования деятельности операторов для
оптимизации управления ресурсами служб поддержки с учетом прогнозируемой
нагрузки

Обучающийся / Student Казаков Михаил Вячеславович

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет информационных
технологий и программирования

Группа/Group М34391

Направление подготовки/ Subject area 01.03.02 Прикладная математика и информатика

Образовательная программа / Educational program Информатика и программирование
2019

Язык реализации ОП / Language of the educational program Русский

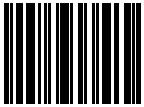
Статус ОП / Status of educational program

Квалификация/ Degree level Бакалавр

Руководитель ВКР/ Thesis supervisor Муравьев Сергей Борисович, кандидат технических
наук, Университет ИТМО, институт прикладных компьютерных наук, доцент
(квалификационная категория "ординарный доцент")

Консультант не из ИТМО / Third-party consultant Якупов Илья Юрьевич, ООО
Яндекс.Технологии, Руководитель группы, кандидат технических наук

Обучающийся/Student

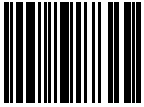
Документ подписан	
Казаков Михаил Вячеславович	
15.05.2023	

(эл. подпись/ signature)

Казаков Михаил
Вячеславович

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Муравьев Сергей Борисович	
15.05.2023	

(эл. подпись/ signature)

Муравьев
Сергей
Борисович

(Фамилия И.О./ name
and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ /
OBJECTIVES FOR A GRADUATION THESIS**

Обучающийся / Student Казаков Михаил Вячеславович

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет информационных технологий и программирования

Группа/Group М34391

Направление подготовки/ Subject area 01.03.02 Прикладная математика и информатика

Образовательная программа / Educational program Информатика и программирование 2019

Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Бакалавр

Тема ВКР/ Thesis topic Разработка системы автоматического планирования деятельности операторов для оптимизации управления ресурсами служб поддержки с учетом прогнозируемой нагрузки

Руководитель ВКР/ Thesis supervisor Муравьев Сергей Борисович, кандидат технических наук, Университет ИТМО, институт прикладных компьютерных наук, доцент (квалификационная категория "ординарный доцент")

Консультант не из ИТМО / Third-party consultant Якупов Илья Юрьевич, ООО Яндекс.Технологии, Руководитель группы, кандидат технических наук

Основные вопросы, подлежащие разработке / Key issues to be analyzed

Для сервиса планирования работы команд в службе поддержки, колл центре и т.д. требуется разработать систему планирования деятельности операторов в команде. Для команды задан временной интервал, который обычно равен 15 минутам. На каждом интервале задан прогноз в условных единицах (числом), причем для каждого оператора известно, сколько условных единиц прогноза он может покрыть за интервал. Горизонт планирования от недели до месяца.

Для каждого оператора есть ограничения, настраиваемые его руководителем. Например, график работы (например, 5/2, выходные в субботу и воскресенье), допустимые часы работы, месячная норма работы, минимальная/максимальная длина смены.

Разработанная система должна составлять расписание, в котором располагаемый ресурс операторов равномерно покрывает прогнозируемую нагрузку и при этом отсутствуют нарушения заданных администратором ограничений.

Форма представления материалов ВКР / Format(s) of thesis materials:

пояснительная записка, презентация, программный код

Дата выдачи задания / Assignment issued on: 03.10.2022

Срок представления готовой ВКР / Deadline for final edition of the thesis 15.05.2023

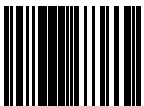
Характеристика темы ВКР / Description of thesis subject (topic)

Тема в области фундаментальных исследований / Subject of fundamental research: нет / not

Тема в области прикладных исследований / Subject of applied research: да / yes

СОГЛАСОВАНО / AGREED:

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Муравьёв Сергей Борисович	
20.03.2023	

(эл. подпись)

Муравьёв
Сергей
Борисович

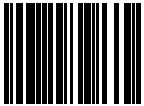
Задание принял к
исполнению/ Objectives
assumed BY

Документ подписан	
Казаков Михаил Вячеславович	
20.03.2023	

(эл. подпись)

Казаков Михаил
Вячеславович

Руководитель ОП/ Head
of educational program

Документ подписан	
Станкевич Андрей Сергеевич	
22.05.2023	

(эл. подпись)

Станкевич
Андрей
Сергеевич

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
SUMMARY OF A GRADUATION THESIS**

Обучающийся / Student Казаков Михаил Вячеславович
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет информационных технологий и программирования
Группа/Group M34391
Направление подготовки/ Subject area 01.03.02 Прикладная математика и информатика
Образовательная программа / Educational program Информатика и программирование 2019
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Бакалавр
Тема ВКР/ Thesis topic Разработка системы автоматического планирования деятельности операторов для оптимизации управления ресурсами служб поддержки с учетом прогнозируемой нагрузки
Руководитель ВКР/ Thesis supervisor Муравьев Сергей Борисович, кандидат технических наук, Университет ИТМО, институт прикладных компьютерных наук, доцент (квалификационная категория "ординарный доцент")
Консультант не из ИТМО / Third-party consultant Якупов Илья Юрьевич, ООО Яндекс. Технологии, Руководитель группы, кандидат технических наук

**ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
DESCRIPTION OF THE GRADUATION THESIS**

Цель исследования / Research goal

Оптимизация управления ресурсами служб поддержки

Задачи, решаемые в ВКР / Research tasks

а) Разработка алгоритма планирования расписаний; б) сравнение реализованного алгоритма с другими решениями; в) внедрение алгоритма в систему.

Краткая характеристика полученных результатов / Short summary of results/findings

Был разработан и внедрен алгоритм планирования ресурсов служб поддержки. Реализованный алгоритм равномерно планирует расписание для команды в службе поддержки. Алгоритм был сравнен с другими решениями. Сравнение алгоритмов показало, что разработанный программный код оптимальнее планирует смены операторов, чем реализованный в системе жадный алгоритм. Также выяснилось, что выбранный алгоритм быстрее сходится, чем другие рассмотренные.

Обучающийся/Student

Документ подписан	
----------------------	--

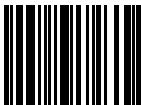
	
Казаков Михаил Вячеславович	
15.05.2023	

(эл. подпись/ signature)

Казаков Михаил
Вячеславович

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Муравьев Сергей Борисович	
15.05.2023	

(эл. подпись/ signature)

Муравьев
Сергей
Борисович

(Фамилия И.О./ name
and surname)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1. Задача планирования дежурств	7
1.1. Планирование дежурств медсестер	7
1.1.1. Использование задачи о рюкзаке	8
1.1.2. Целочисленное линейное и квадратичное программирование	9
1.1.3. Жадный алгоритм	12
1.1.4. Эволюционный алгоритм	13
1.2. Планирование дежурств операторов	16
Выводы по главе 1	18
2. Алгоритм планирования	19
2.1. Оценка качества индивидуума	19
2.2. Алгоритм поиска оптимального индивидуума	24
2.2.1. Генераторы смен	25
2.2.2. Генераторы перерывов	28
2.2.3. Операторы мутации	30
Выводы по главе 2	32
3. Реализация и апробация алгоритма планирования	33
3.1. Архитектура программного кода планирования смен	33
3.1.1. Программная реализация эволюционного алгоритма	33
3.1.2. Ускорение эволюционного алгоритма	37
3.2. Сравнение алгоритмов планирования	41
3.2.1. Сравнение с решением задачи планирования дежурств медсестер	41
3.2.2. Применение других эволюционных алгоритмов	44
Выводы по главе 3	48
ЗАКЛЮЧЕНИЕ	49
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	51

ВВЕДЕНИЕ

Из года в год требования к сервисам растут. Уже сегодня от многих сервисов ожидается круглосуточная бесперебойная работа. Такие же требования предъявляются в том числе и к службам поддержки. В таких сервисах довольно важно распределить ресурсы операторов. Например, в час пик на смену необходимо выйти большему числу исполнителей, чем ночью.

Поэтому на любом таком предприятии рано или поздно появляются как задача планирования, так и задача прогнозирования нагрузки. Эти задачи может решать вручную руководитель команды операторов, но таким способом планирование может затянуться на несколько часов, а то и дней. Автоматизация процесса планирования оптимизирует время руководителей, а также улучшит качество покрытия прогноза, поэтому задача актуальна и имеет практическое значение. Целью данной работы будем считать оптимизацию управления ресурсами служб поддержки.

Задачи автоматизации решаются разработкой программного обеспечения. Одной из задач данной работы является разработка алгоритма, автоматизирующего процесс планирования смен. Важно, что алгоритм может контролировать начало смены сотрудника и ее длину, поэтому команды со свободным графиком не рассматриваются. Также важно, что для поступающей нагрузки установлен некий максимальный уровень отклика, то есть работа не может быть отложена.

Под аналогичные требования к алгоритму попадает также и работа больниц. Больные не могут ждать часами, пока к ним придет медсестра. Также в каждый момент времени в больнице должна находиться хотя бы одна медсестра, на случай форс-мажора. Поэтому разработанный алгоритм должен быть сравнен с решениями задачи планирования работы больниц.

Для расписаний больниц уже сформулирована модель (en. NSP, nurse scheduling problem или nurse rostering problem, задача планирования дежурств медсестер), которая уже довольно долго изучается (более 50 лет [4]). Под разные ограничения человечество придумало множество решений: некоторые находят глобальный оптимум, некоторые локальный. Тем не менее было доказано, что планирование смен медсестер в больнице — NP-трудная задача [8]. То есть поиск глобального оптимума займет неприемлемо много времени и необходимо оптимизировать алгоритм для поиска локального оптимума.

Тем не менее, решаемая в данной работе задача отличается от задачи планирования дежурств медсестер. В задаче планирования дежурств медсестер обычно рассматривают только три шаблона смены — ночь, утро или вечер. В то время как в данной работе рассматриваются такие расписания, в которых число шаблонов смен достигает сотен, причем для разных операторов эти шаблоны разные. Более того, в данной работе требуется спланировать перерывы операторов, что не входит в условия задачи планирования дежурств медсестер.

В первой главе дадим определение задаче планирования дежурств медсестер, а также определим решаемую в данной работе задачу. Также в первой главе существующие решения как для больниц, так и для служб поддержки. Далее, во второй главе, рассмотрим реализованный алгоритм, обсудим его плюсы и минусы. Рассмотрим возможные подходы к реализации алгоритма и выберем лучший. В завершение, в третьей главе обсудим, как реализовать алгоритм на практике, а также сравним полученный алгоритм с другими решениями.

ГЛАВА 1. ЗАДАЧА ПЛАНИРОВАНИЯ ДЕЖУРСТВ

1.1. Планирование дежурств медсестер

Человечество не первый раз решает задачи планирования смен для работников. Самая похожая задача на рассматриваемую в данной работе — это nurse scheduling problem (NSP). У этой задачи много различных постановок, тем не менее они обычно не сильно отличаются. Рассмотрим постановку NSP:

- а) день состоит из трех, возможно пересекающихся, временных отрезка (обычно, утро, вечер и ночь), которые далее будем называть шаблонами смен;
- б) дана команда медсестер, каждая из медсестер может иметь какие-то навыки, предпочтения, должность;
- в) дан горизонт планирования, на который нужно спланировать смены медсестер;
- г) необходимо для каждой медсестры спланировать расписание смен (начало и длительность каждой смены определяется шаблоном).

Причем на полученное расписание накладываются некоторые ограничения. В разных работах [2–4, 10, 12] ограничения предлагаются разные, рассмотрим некоторые примеры:

- а) в день медсестра работает не более одной смены;
- б) максимальное число смен в неделю для каждой медсестры;
- в) для каждого шаблона смены задано минимальное число медсестер определенной должности;
- г) для каждой должности задана производительность в условных единицах, и для каждого шаблона смены задан прогноз работы в тех же условных единицах;
- д) минимальное и/или максимальное число последовательных рабочих/выходных дней;
- е) различия в ночных и дневных шаблонах смен: например, более строгое ограничение на число смен в неделю, если хотя бы одна из них ночная;
- ж) для каждого шаблона смены задан набор навыков, которые имеют только определенные медсестры — для каждого навыка в шаблоне смены должно быть ненулевое число смен медсестер с этим навыком;
- и) для каждой пары медсестер задана совместимость — некоторые пары в совокупности работают лучше, некоторые хуже.

Для решения такой проблемы существует множество решений: некоторые находят глобальный оптимум, некоторые — локальный (если в постановке задачи заданы мягкие ограничения — функции, которые нужно минимизировать или максимизировать). Рассмотрим некоторые решения, а также их плюсы и минусы.

1.1.1. Использование задачи о рюкзаке

Рассмотрим один из способов разделить большую задачу на две задачи поменьше [10]. Перед тем как описать способ, уточним постановку задачи планирования дежурств медсестер:

- а) для каждого шаблона смены известен один из типов — ночной шаблон или дневной шаблон смены;
- б) для каждой медсестры задан тип договора t ;
- в) для каждого типа договора задано d_t и e_t — сколько дневных и ночных смен в неделю может быть назначено на медсестру по договору t соответственно;
- г) за неделю медсестра может работать либо только в ночные смены, либо только в дневные;
- д) в команде есть запасные медсестры, на которых можно назначать смены, если основной команды не хватает;
- е) для каждого шаблона смены задано минимальное число назначенных медсестер;
- ж) число использованных запасных медсестер необходимо минимизировать.

Просуммируем по всем дневным и ночным шаблонам смены минимальное число назначенных медсестер. Пусть D, E — суммарное число необходимых медсестер для дневных и ночных шаблонов смен соответственно. Также пусть y_t, N_t, T — число медсестер с договором типа t , назначенных на ночную смену, число медсестер с договором типа t и число типов договоров соответственно. Тогда запишем ограничения следующим образом:

$$\sum_{t=1}^T e_t y_t \geq E \quad (1)$$

$$\sum_{t=1}^T d_t (N_t - y_t) \geq D \quad (2)$$

$$\forall t : y_t \leq N_t \quad (3)$$

$$\forall t : y_t \in \mathbb{Z}_+ \quad (4)$$

Применив преобразования к неравенству (2), получим следующее

$$\sum_{t=1}^T d_t y_t \leq \sum_{t=1}^T N_t d_t - D \quad (5)$$

Заметим, что в неравенствах (1) и (5) правая часть постоянна, так как изменяется и оптимизируется лишь параметр y_t . Иными словами, из правых частей выражения неравенств (1) и (5) можно тривиально посчитать, имея входные данные.

Далее заметим, что d_t можно считать весом предмета типа t в контексте задачи о рюкзаке, e_t — стоимостью, а y_t — числом предметов типа t в рюкзаке. Таким образом, получим задачу об ограниченном рюкзаке, решение которой будет максимизировать левую часть (1). Если решение подберет недостаточно медсестер, добавим в команду несколько запасных медсестер и перезапустим алгоритм. Пусть C_i — команда на i -й итерации, тогда получим следующее решение задачи:

- а) решить задачу о рюкзаке для команды C_i , получив $M_i = \max_{y_{i,t}} \sum_{t=1}^T e_{i,t} y_{i,t}$;
- б) если $M_i \geq E$, вернуть $y_{i,t}$;
- в) иначе добавить в команду $E - M_i$ запасных медсестер и перейти к шагу (а).

Таким образом, получим y_t , с помощью чего для каждой медсестры можно определить тип смен, в которые она будет работать. Например, это можно делать жадным алгоритмом. Далее можно решать задачу независимо для дневных и ночных смен, если это позволяют ограничения.

1.1.2. Целочисленное линейное и квадратичное программирование

Рассмотрим одно из решений задачи планирования дежурств медсестер [16]. Заметим, что назначение на шаблон смены можно представить в ви-

де логической переменной. Пусть $a_{i,j} = 1$, если i -я медсестра назначена на j -й шаблон смены, иначе $a_{i,j} = 0$. Тогда большинство ограничений задачи можно представить с помощью линейной комбинации. Например, с помощью следующего равенства можно задать ограничение на ровно одну смену в первый день планируемого периода:

$$a_{i,1} + a_{i,2} + a_{i,3} = 1. \quad (6)$$

Аналогично, зададим ограничение на минимальное число медсестер, имеющих навык g . Для простоты положим, что в первый шаблон смены должна быть хотя бы одна медсестра с навыком g . Пусть $G = \{i \mid \text{медсестра } i \text{ имеет навык } g\}$, тогда зададим ограничение для первого шаблона смены следующим образом

$$\sum_{i \in G} a_{i,1} \geq 1. \quad (7)$$

В разных постановках задачи к расписанию предъявляются разные требования. Иногда достаточно найти какое-то допустимое расписание, но часто при этом необходимо найти «хорошее» решение. Формально, необходимо оптимизировать некоторую метрику в пространстве допустимых решений. Например, в задаче часто заданы мягкие ограничения — функции, которые необходимо минимизировать для решения. Если таких ограничений несколько, то необходимо их как-то агрегировать, например просуммировать.

Например, для каждого шаблона смены и медсестры может быть известен штраф, если у организации есть какие-то сведения о продуктивности медсестры по каждому периоду. Аналогичное ограничение может быть задано при учете предпочтений медсестер. Пусть $c_{i,j}$ — штраф за назначение i -й медсестры на j -й шаблон смены (штраф может быть отрицательным). Также пусть S^T и K — число шаблонов смен и число медсестер соответственно, тогда зададим ограничение следующим образом

$$\sum_{i=1}^K \sum_{j=1}^{S^T} c_{i,j} a_{i,j} \rightarrow \min \quad (8)$$

Рассмотренные выше ограничения описывают задачу целочисленного линейного программирования. Такую задачу часто решают с помощью обыч-

ной задачи линейного программирования. Для поиска целочисленного решения часто используют метод ветвей и границ [5].

Рассмотрим метод ветвей и границ на примере поиска минимума функции. Введем понятие релаксированной задачи линейного программирования. В изначальной задаче $a_{i,j} \in \{0, 1\}$, в релаксированной же $a_{i,j} \in [0, 1]$. Тогда релаксированную задачу можно решать методами обычного линейного программирования.

Рассмотрим как с помощью релаксированной задачи решить задачу целочисленного линейного программирования. Для пространства \mathcal{Q} и оптимизируемой функции f поиска задается функция $\Phi(\mathcal{Q})$ — минимум множества $f(\mathcal{Q})$ для релаксированной задачи. Алгоритм инициализируется стартовым пространством поиска \mathcal{Q}_0 и каждую итерацию поддерживает семейство рассматриваемых множеств $\hat{\mathcal{Q}}$. Также алгоритм поддерживает лучшее решение a^b , итеративно улучшая решение. Будем считать, что все рассматриваемые множества являются гиперпрямоугольниками. На каждой итерации необходимо выполнить следующие шаги:

- а) выбрать множество \mathcal{Q}_i из $\hat{\mathcal{Q}}$ и удалить его из семейства;
- б) если $\Phi(\mathcal{Q}_i)$ хуже, чем a^b , то вернуться к шагу (а);
- в) разделить множество \mathcal{Q}_i на \mathcal{Q}_i^1 и \mathcal{Q}_i^2 ;
- г) если $\Phi(\mathcal{Q}_i^k)$ — целочисленное решение оптимальнее a^b , то обновить a^b для $k \in \{1, 2\}$;
- д) добавить \mathcal{Q}_i^1 и \mathcal{Q}_i^2 в $\hat{\mathcal{Q}}$.

Для реализации шага (а) можно псевдослучайно выбирать множество из семейства. Также можно вычислять оценки на целочисленный минимум и выбирать множество с самой точной оценкой. Для реализации шага (в) можно фиксировать значение $a_{i,j}$ для каких-то i и j , которые еще не были зафиксированы. Например, для \mathcal{Q}_i^1 добавить ограничение $a_{i,j} = 0$, а для \mathcal{Q}_i^2 — ограничение $a_{i,j} = 1$.

Иногда оптимизируемая функция нелинейна и линейное программирование не применимо. Тем не менее, такую функцию иногда можно выразить квадратичной функцией — такую функцию можно оптимизировать методами целочисленного квадратичного программирования.

Рассмотрим плюсы данного подхода:

- а) алгоритм можно параллелизовать и даже реализовать для исполнения в распределенной системе, например, разделяя Q для каждого потока;
- б) алгоритм можно преждевременно остановить, если полученное решение устраивает пользователя.

А также перечислим его минусы:

- а) задача экспоненциально растет от числа переменных, а их часто очень много;
- б) алгоритм может работать дольше обычного перебора, если решение существует только в релаксированной задаче.

1.1.3. Жадный алгоритм

Рассмотрим жадный алгоритм [2], который делает очень мало предположений о структуре задачи. Единственное предположение, которое алгоритм делает — решение можно численно оценить. Основная идея алгоритма — итеративно добавлять самую лучшую по какому-то критерию смену так, что полученное расписание удовлетворяет ограничениям. Сам критерий зависит от точной постановки задачи, но его можно считать независимо от жадного алгоритма. Поэтому данный алгоритм подходит для любой постановки задачи планирования, если расписание можно численно оценить.

Рассмотрим один из примеров использования данного алгоритма [2]. Будем постепенно добавлять смену на каждый шаблон смены, начиная с первого шаблона в планируемом периоде. Добавлять смену к шаблону будем пока полностью не покроем ограничения по медсестрам, либо пока не просмотрим всех доступных медсестер. Для итеративного выбора лучшей медсестры на каждую смену будем использовать критерий честности k_i по отношению к медсестрам. Зададим критерий k_i для медсестры i следующим образом: пусть t , $S_{t,i}^c$ — тип смены (утренняя, вечерняя или ночная) и число смен типа t у медсестры i за последние два месяца, тогда

$$k_i = S_{t,i}^c \quad (9)$$

Тогда алгоритм назначит смену на ту медсестру, которая меньше всего работала в такой тип смены за последние два месяца. Рассмотрим плюсы данного подхода:

- а) алгоритм можно реализовать для любой задачи, в которой решение можно численно оценить.

А также перечислим его минусы:

- а) алгоритм работает довольно долго, так как на каждой итерации пересчитывает значение штрафа для каждой допустимой смены, алгоритм нельзя остановить в произвольный момент времени и получить какое-то допустимое промежуточное решение;
- б) часто покрытие может быть сильно улучшено небольшими изменениями расписания, например, смену старшей медсестры можно переместить из шаблона с избытком медсестер в шаблон с недостатком медсестер.

Далее рассмотрим подход, который решает пункт (б) из списка минусов данного алгоритма.

1.1.4. Эволюционный алгоритм

Эволюционные алгоритмы основаны на небольших изменениях решения так, чтобы оно приблизилось к оптимуму. Изменения должны быть маленькими, так как большое изменение может привести к «перепрыгиванию» оптимума. Такие изменения обычно называют мутациями. Оптимизируемую функцию в эволюционных алгоритмах обычно называют функцией приспособленности.

Функция приспособленности может быть как однокритериальная, так и многокритериальная. Часто при решении задачи необходимо оптимизировать несколько критериев. В задаче может быть описано несколько мягких ограничений, например, предпочтения медсестер и приоритеты больниц — так и появляются многокритериальные функции приспособленности. Заметим, что в многокритериальной оптимизации не все решения сравнимы. Для этого определяют отношение доминирования \prec . Пусть $f(S)$ — функция приспособленности, а $f(S)_i$ — её i -й компонент, тогда определим отношение доминирования следующим образом

$$f(S) \prec f'(S) \leftrightarrow \forall i : f(S)_i \leq f'(S)_i, \exists i : f(S)_i < f'(S)_i \quad (10)$$

Также определим множество Парето ρ^s и фронт Парето ρ^f :

$$\rho^s = \{S \mid \forall S' \neq S : f(S) \prec f(S')\} \quad (11)$$

$$\rho^f = f(\rho^s) \quad (12)$$

Многокритериальная оптимизация подразумевает поиск множества Парето или его приближение. Конечно, при многокритериальной оптимизации можно использовать скаляризацию — агрегировать критерии и решать задачу для однокритериальной функции приспособленности. Тем не менее, такие решения не находят некоторые решения на невыпуклом фронте Парето. Также метод скаляризации может давать плохую сходимость, так как он не имеет полной информации о каждом критерии функции приспособленности.

Рассмотрим пример однокритериального эволюционного алгоритма $(N + K)$ [11]. Алгоритм поддерживает популяцию из K решений. Популяция для первой итерации генерируется специальным генератором решений, который предоставляет пользователь. Каждую итерацию алгоритм выбирает N , возможно, повторяющихся решений из текущей популяции и применяет к ним заданные пользователем операторы мутации. Получив новые решения, алгоритм их оценивает с помощью функции приспособленности и переходит к стадии отбора. На стадии отбора алгоритм объединяет K изначальных решений и N полученных, затем сортирует их по значению функции приспособленности, после чего берет K лучших решений. Такая стадия отбора возможна, так как функция приспособленности однокритериальная, то есть все решения сравнимы и сортировка для популяции определена.

Рассмотрим пример многокритериального эволюционного алгоритма ESPEA [6]. Данный алгоритм поддерживает архив недоминируемых решений, то есть является приближением множества Парето. На каждой итерации к случайному решению из архива применяется оператор мутации. Полученное решение добавляется обратно в архив.

Так как ресурсы вычислительных машин ограничены, алгоритму необходимо поддерживать архив конечного размера. Обычно, считают, что в архиве содержится не более N решений. Когда архив переполняется, необходимо удалить какое-то решение. Для этого, обычно, используют какой-то критерий. Тогда решение с худшим значением удаляется из архива.

В алгоритме ESPEA [6] предлагается критерий «энергии». Во-первых, из архива удаляются доминируемые решения. Во-вторых, если все решения недоминируемы, из архива удаляется решение с наибольшим значением энергии E_i . Пользователь задает веса w_i — насколько каждое решение важно для

него. Для функции приспособленности f с K критериями и решения S_i определим энергию следующим образом:

$$E_i = w_i \sum_{j \neq i} w_j \left(\sum_{k=1}^K (f(S_i)_k - f(S_j)_k)^2 \right)^{-0.5} \quad (13)$$

Рассмотрим пример еще одного многокритериального эволюционного алгоритма NSGA-II [1]. Данный алгоритм работает аналогично предыдущему. Различается лишь способ фильтрации архива. Во-первых, в отличие от ESPEA, в архиве поддерживаются доминируемые решения. Во-вторых, решения сортируются по паре критериев (R_i, C_i) , то есть решения сравниваются по критерию R_i , а при прочих равных по критерию C_i . Опишем следующий итеративный процесс: на каждой итерации будем удалять решения из множества Парето. Тогда R_i — номер итерации, на которой удалено i -е решение. Такой алгоритм называют недоминирующей сортировкой, для которого существует реализация со сложностью $\mathcal{O}(N \log^{K-1} N)$ [7]. Пусть решения S_i^j отсортированы по $f(S_i)_j$, тогда критерий C_i опишем следующей формулой:

$$C_i = \sum_{j=1}^K \frac{f(S_{i+1}^j)_j - f(S_{i-1}^j)_j}{\max_i f(S_i)_j - \min_i f(S_i)_j} \quad (14)$$

Также, помимо описанных, ранее было разработано довольно много эволюционных алгоритмов, например, SPEA2 [17], PESA-II [15], NSGA-III [9] и RNSGA-II [14]. Данные эволюционные алгоритмы реализованы, например, в программной платформе jMetal [13].

В разных работах описаны разные операторы мутации. Например, можно представить расписание медсестры в виде последовательности D длины D^l , где D^l — число дней в планируемом периоде, а D_i — назначение медсестры на первую, вторую или третью смену. Если $D_i = 0$, будем считать, что у медсестры выходной в i -й день. Тогда можно реализовать следующий оператор мутации: выбираем случайную медсестру, случайный день и случайно меняем смену на другую [3].

Рассмотрим плюсы эволюционного подхода:

- а) алгоритм улучшает решение небольшими шагами, из-за чего алгоритм можно остановить в любой момент и получить решение;

- б) алгоритм можно расширять, добавлять новые ограничения и свойства операторов, сильно не меняя структуру алгоритма.

А также перечислим его минусы:

- а) алгоритм часто не достигает глобального оптимума и «застревает» в локальном оптимуме;
- б) часто для достижения оптимума алгоритму требуется довольно много времени.

1.2. Планирование дежурств операторов

Решаемая в данной работе задача несколько отличается от NSP. Рассмотрим постановку задачи:

- а) день разделен на непересекающиеся отрезки времени одинаковой длины (обычно, 15 минут), которые далее будем называть шагами;
- б) на каждый шаг задан прогноз работы в условных единицах;
- в) дана команда операторов службы поддержки, каждый оператор имеет различные ограничения на смены и производительность (сколько условных единиц прогноза выполняет за шаг);
- г) на каждый шаг можно назначить несколько операторов: максимальные по включению отрезки шагов, назначенные на одного оператора, будем называть сменой;
- д) во время смены могут быть помечены специальные шаги, в течение которых производительность оператора равна нулю: максимальные по включению отрезки таких шагов будем называть перерывами;
- е) дан горизонт планирования, длина которого — от недели до месяца;
- ж) необходимо для каждого оператора спланировать расписание смен и перерывы, равномерно покрывая спрогнозированную нагрузку и удовлетворяя ограничениям операторов.

Также у каждого оператора есть дополнительные свойства:

- а) часовой пояс проживания оператора;
- б) норма работы за период планирования, например, 168 часов;
- в) отсутствия на какой-то период, например, отпуск;
- г) периоды начала смены, заданные в часовом поясе оператора, например, 09:00-11:00 UTC и 12:00-12:30 UTC;

- д) рабочий график, который задается последовательными рабочими и выходными днями, а также первым рабочим днем, например, 5/2, начиная с первого июня 2023-го года;
- е) ночной период, заданный в часовом поясе оператора, например, 02:00-06:00 MSK.

Также рассмотрим ограничения, которые накладываются на смены операторов:

- а) максимальная длина смены — восемь часов, длина смены положительна;
- б) дни начала всех смен одного оператора в его часовом поясе различны;
- в) каждая смена должна пересекаться с горизонтом планирования;
- г) суммарная длина всех смен должна быть равна норме;
- д) смена оператора не должна пересекаться с отсутствием;
- е) между сменами должен быть отдых (последовательные шаги, на которые оператор не назначен) — хотя бы 12 часов;
- ж) смена оператора может начинаться только в заданные периоды начала смены, причем:
 - 1) смена не может начинаться в выходной по рабочему графику;
 - 2) смена не может начинаться в ночной период.
- и) длины смен конкретного оператора не должны сильно различаться.

Введем определение рабочего периода для оператора — максимальный по включению подотрезок смены оператора, каждый шаг которого не пересекается с перерывами. Также разделим перерывы на два типа: обычный (15 минут) и обеденный (30 минут), другие перерывы планировать не планируется. Рассмотрим ограничения, которые накладываются на перерывы в течение одной смены:

- а) для каждого типа перерыва и длины смены задано число перерывов, которые необходимо добавить;
- б) первый тип перерыва в смене — обычный;
- в) длина любого рабочего периода в смене должна быть не менее одного часа и не более трех часов;
- г) перерыв не должен выходить за пределы смены.

Потенциально, конкретные числа могут настраиваться на каждого оператора, но в существующей системе на данный момент реализована настройка только части параметров.

Выводы по главе 1

В данной главе была дана постановка задачи планирования как в больницах, так и в службах поддержки. Для задачи планирования дежурств медсестер были описаны решения, которые уже были предложены ранее. Рассмотрим отличия поставленной задачи от NSP:

- а) в NSP часто требуется полное покрытие прогноза, а в поставленной задаче требуется равномерно покрыть прогноз (возможно, не полностью);
- б) в отличие от NSP, нужно спланировать перерывы операторов;
- в) в то время как в NSP всего 3 шаблона смен, в поставленной задаче их число может достигать сотен ($24 \cdot \frac{60}{15} = 96$ шагов за день, длина каждой смены до 24 шагов);
- г) шаблоны смен NSP пересекаются отрезком небольшой длины (например, час), в то время как в поставленной задаче различные шаблоны смен могут быть равными с точностью до одного шага.

Таким образом, необходимо разработать алгоритм, который будет учитывать все пункты, описанные выше, так как задача отличается от NSP. Заметим, что использование квадратичного программирования не подходит в качестве решения, так как для сведения потребуется конструировать огромную матрицу. Также при решении задачи жадным алгоритмом расписание будет хорошо строиться на ранних этапах и будет иметь трудности на поздних этапах. Например, жадный алгоритм всегда ставит смену максимальной длины на ранних этапах, из-за чего норма может не покрыться, так как она не делится на длину максимальной смены. Поэтому далее рассмотрим решение, основанное на эволюционном алгоритме.

ГЛАВА 2. АЛГОРИТМ ПЛАНИРОВАНИЯ

2.1. Оценка качества индивидуума

Для начала необходимо обозначить способ оценки полученного алгоритмом расписания, то есть определить функцию приспособленности, которую далее будет оптимизировать алгоритм. Будем считать, что область поиска решения почти не ограничена, то есть некорректные решения тоже необходимо оценить. Конкретнее, оценка не будет предоставлена только тем решениям, которые невозможно представить в реальном мире. Например, расписание, в котором есть смены отрицательной длины.

Представлять оценку числом довольно неэффективно, так как из-за этого теряется информация о других оптимизирующихся метриках. Поэтому результатом разработанной функции приспособленности будет вектор чисел с плавающей точкой. Слишком большая длина результирующего вектора функции приспособленности также ухудшит сходимость алгоритма, так как множество Парето будет слишком большим. Поэтому необходимо выбрать не слишком много координат и не слишком мало. Возьмем следующие размерности для функции приспособленности:

- а) равномерность покрытия прогноза;
- б) качество удовлетворения строгим ограничениям;
- в) качество удовлетворения мягким ограничениям.

Рассмотрим метрику покрытия прогноза. Будем считать, что равномерность подразумевает относительное равенство покрытия прогноза на каждом шагу. Это значит, что отношение проделанной за шаг работы к прогнозу работы на этот шаг должно быть примерно одинаковым. В качестве метрики равномерности списка чисел будем использовать несмещённую дисперсию. Для нормировки данной метрики поделим ее на среднее, получив при этом индекс дисперсии.

Заметим, что с такой метрикой оптимум будет находиться при одинаковом относительном покрытии прогноза, в том числе и на нулевом покрытии прогноза. Чтобы нулевое покрытие прогноза не было оптимальным, нужно награждать алгоритм за приближение фактической работы к прогнозируемой. Для достижения этого результата домножим индекс дисперсии на процент прогноза, который еще необходимо покрыть.

Пусть A_i — сумма производительностей операторов, назначенных на i -й шаг, а F_i — прогноз для i -го шага, $D(X)$ и $E(X)$ — несмещенная дисперсия и среднее списка X . Тогда метрика покрытия прогноза P^f будет представлена следующим образом:

$$K_i = \frac{A_i}{F_i}; \quad H_i = \frac{\max(0, F_i - A_i)}{F_i} \quad (15)$$

$$P^f = \left(1 + \frac{D(K)}{E(K)}\right) (1 + E(H)) - 1 \quad (16)$$

Далее разделим ограничения на мягкие и строгие. Зададим следующие ограничения как мягкие, а остальные ограничения будем считать строгими:

- а) длины смен конкретного оператора не должны сильно различаться.

Для равномерной длины смен не получится добавить строгое ограничение, так как норма может не делиться на число рабочих дней в периоде планирования. Поэтому данное ограничение было реализовано как мягкое.

Заметим, что алгоритму также необходимо понимать, насколько плохо решение, не удовлетворяющее строгим ограничениям. Поэтому для каждого строгого ограничения необходимо дать оценку, насколько сильно оно нарушается.

Здесь и далее будем считать, что смены в расписании оператора упорядочены по моменту начала смены (для смен, начинающихся в одно время порядок неопределен). Пусть

- а) K — число операторов в команде;
- б) S_k — число смен в расписании оператора k ;
- в) $S_{k,i}^s$ — момент начала i -й смены в расписании оператора k ;
- г) $S_{k,i}^e$ — момент конца i -й смены в расписании оператора k ;
- д) $S_{k,i}^l$ — длина i -й смены в расписании оператора k : $S_{k,i}^l = S_{k,i}^e - S_{k,i}^s$.

Для каждого ограничения запишем формулу, по которой будет считаться штраф расписания оператора k :

- а) максимальная длина смены — восемь часов, длина смены положительна: пусть m_k , M_k — минимальная и максимальная длина смены для оператора соответственно, тогда

$$P_{1,k}^h = \sum_{i=1}^{S_k} \begin{cases} \frac{m_k - S_{k,i}^l}{S_k m_k}, S_{k,i}^l < m_k \\ \frac{S_{k,i}^l - M_k}{S_k M_k}, S_{k,i}^l > M_k \\ 0, \text{ иначе} \end{cases} ; \quad (17)$$

б) дни начала всех смен одного оператора в его часовом поясе различны: пусть $D_k(i)$ — день начала смены i оператора k в его часовом поясе, тогда

$$P_{2,k}^h = \begin{cases} 0, S_k = 0 \\ 1 - \frac{|\{D_k(1), D_k(2), \dots, D_k(S_k)\}|}{S_k}, \text{ иначе} \end{cases} ; \quad (18)$$

в) каждая смена должна пересекаться с горизонтом планирования: пусть I_k — число смен, которые не пересекаются с горизонтом планирования, тогда

$$P_{3,k}^h = \begin{cases} 0, S_k = 0 \\ \frac{I_k}{S_k}, \text{ иначе} \end{cases} ; \quad (19)$$

г) суммарная длина всех смен должна быть равна норме: пусть N_k — норма работы за период планирования для оператора k , тогда

$$P_{4,k}^h = \left| 1 - \frac{\sum_{i=1}^{S_k} S_{k,i}^l}{N_k} \right| ; \quad (20)$$

д) смена оператора не должна пересекаться с отсутствием: пусть $A_k(i)$ — длина пересечения i -й смены оператора k со всеми отсутствиями, тогда

$$P_{5,k}^h = \begin{cases} 0, S_k = 0 \\ \frac{\sum_{i=1}^{S_k} A_k(i)}{\sum_{i=1}^{S_k} S_{k,i}^l}, \text{ иначе} \end{cases} ; \quad (21)$$

е) между сменами должен быть отдых (последовательные шаги, на которые оператор не назначен) — хотя бы 12 часов: пусть R_k — количество

времени, которое необходимо оператору для отдыха между сменами, тогда

$$P_{6,k}^h = \sum_{i=1}^{S_k-1} \frac{\max(0, R_k - (S_{k,i+1}^s - S_{k,i}^e))}{R_k S_k}; \quad (22)$$

ж) смена оператора может начинаться только в заданные периоды начала смены: пусть B_k — множество отрезков из планируемого периода, в которые может начинаться смена оператора k , тогда

$$P_{7,k}^h = \begin{cases} 0, S_k = 0 \\ \frac{|\{i | S_{k,i}^s \notin B_k\}|}{S_k}, \text{ иначе} \end{cases}; \quad (23)$$

и) длины смен конкретного оператора не должны сильно различаться: пусть $D(S_k^l)$ — индекс дисперсии длин смен оператора k , тогда

$$P_{1,k}^s = D(S_k^l). \quad (24)$$

Ограничения на перерывы локальны для каждой смены. Для каждого ограничения на перерывы оператора k и смены i в его расписании запишем формулу, по которой будет считаться штраф:

а) для каждого типа перерыва и длины смены задано число перерывов, которые необходимо добавить: пусть B^t , $B_{i,j,k}^t$, $B_{j,k}^t$ — число типов перерывов, число перерывов типа j в смене i оператора k и заданное число перерывов типа j для оператора k соответственно, тогда

$$P_{1,k,i}^{b,h} = \sum_{j=1}^{B^t} \frac{|B_{i,j,k}^t - B_{j,k}^t|}{B_{j,k}^t B^t}; \quad (25)$$

б) первый тип перерыва в смене — обычный:

$$P_{2,k,i}^{b,h} = \begin{cases} 1, \text{ ограничение соблюдено} \\ 0, \text{ иначе} \end{cases}; \quad (26)$$

в) длина любого рабочего периода в смене должна быть не менее одного часа и не более трех часов: пусть $W_{j,i,k}^l$, $W_{i,k}^c$, W_k^m , W_k^M — длина j -го рабочего периода смены i оператора k , число рабочих периодов в смене i

оператора k , минимальная и максимальная длина рабочего периода для оператора k соответственно, тогда

$$P_{3,k,i}^{b,h} = \sum_{j=1}^{W_{i,k}^c} \begin{cases} \frac{W_k^m - W_{j,i,k}^l}{W_{i,k}^c W_k^m}, W_{j,i,k}^l < W_k^m \\ \frac{W_{j,i,k}^l - W_k^M}{W_{i,k}^c W_k^M}, W_{j,i,k}^l > W_k^M \\ 0, \text{ иначе} \end{cases} ; \quad (27)$$

г) перерыв не должен выходить за пределы смены: пусть $B_{i,k}$, $B_{j,i,k}^s$, $B_{j,i,k}^l$ — число перерывов в смене i оператора k , длина пересечения со сменой i j -го перерыва в смене i оператора k и длина j -го перерыва в смене i оператора k соответственно, тогда

$$P_{4,k,i}^{b,h} = \sum_{j=1}^{B_{i,k}} \frac{B_{j,i,k}^l - B_{j,i,k}^s}{B_{i,k} B_{j,i,k}^l}. \quad (28)$$

Слишком много размерностей в функции приспособленности (в нашем случае, порядка десяти) приведет к тому, что разнообразие популяции будет сильно изменяться, при изменении одного критерия. Поэтому необходимо агрегировать полученные штрафы, уменьшив число размерностей в функции приспособленности. Посчитаем среднее для каждой полученной группы штрафов:

$$P_j^h = \sum_{k=1}^K \frac{P_{j,k}^h}{K}; \quad (29)$$

$$P_j^s = \sum_{k=1}^K \frac{P_{j,k}^s}{K}; \quad (30)$$

$$P_{j,k}^{b,h} = \sum_{i=1}^{S_k} \frac{P_{j,k,i}^{b,h}}{S_k}; \quad (31)$$

$$P_j^{b,h} = \sum_{k=1}^K \frac{P_{j,k}^{b,h}}{K}. \quad (32)$$

Далее определим положительные веса для каждой группы штрафов: $w_1^h, w_2^h, \dots; w_1^s, w_2^s, \dots; w_1^{b,h}, w_2^{b,h}, \dots$.

Запишем формулу для оценки удовлетворимости строгим ограничениям P^h и мягким ограничениям P^s :

$$P^h = \left(\sum_{j=1} w_j^h P_j^h \right) + \left(\sum_{j=1} w_j^{b,h} P_j^{b,h} \right); \quad (33)$$

$$P^s = \sum_{j=1} w_j^s P_j^s. \quad (34)$$

Несмотря на то, что алгоритм может возвращать набор решений, каждое из которых хорошо по-своему, требуется автоматически получить единственное решение. Поэтому определим численную оценку решения:

$$P = (1 + P^f) (1 + P^h) (1 + P^s). \quad (35)$$

Причем приоритет будем отдавать решениям, для которых $P^h = 0$. Произведение штрафов различных ограничений было выбрано для того, чтобы лучшим решением оказалось среднее по всем параметрам решение, даже если подобрана неправильная нормировка.

2.2. Алгоритм поиска оптимального индивидуума

Далее рассмотрим алгоритм, который будет оптимизировать поставленную функцию приспособленности. В прошлой главе было замечено, что и решение квадратичной системы уравнений, и поиск глобально оптимального решения займет слишком много времени. Поэтому было принято решение использовать эволюционные алгоритмы, которые двигаются к некоторому локальному минимуму, немного изменяя решение.

Так как оптимизировать придется несколько критериев, то был выбран многокритериальный алгоритм ESPEA [6]. Если наивно реализовывать ESPEA, то в популяции будет находиться слишком мало индивидуумов, удовлетворяющих строгим ограничениям. Это происходит из-за того, что многокритериальные алгоритмы поддерживают разнообразие популяции, а равенство нулю для одной из компонент функции приспособленности — это слишком малое разнообразие. Запишем модификации, которые были применены к исходному алгоритму для исправления этой проблемы:

- а) необходимо поддерживать половину популяции удовлетворяющими строгим ограничениям;

- б) необходимо поддерживать лучшее решение в смысле метрики P , описанной выше.

2.2.1. Генераторы смен

Алгоритм ESPEA, как и любой эволюционный алгоритм, предполагает мутации (небольшие локальные изменения) решения на каждой итерации. Но перед мутациями алгоритму необходимо задать некую стартовую точку — начальное решение, с помощью которого алгоритм будет двигаться к оптимуму. Алгоритм может начать с пустого расписания, но в таком случае потребуются огромное число итераций для получения решения, которое удовлетворяет строгим ограничениям. Поэтому была реализована генерация решений, которые не только удовлетворяют строгим ограничениям, но и довольно хорошо оптимизированы под мягкие ограничения. Далее рассмотрим генераторы смен без перерывов, а перерывы будем генерировать независимо для каждой смены. Перечислим разработанные генераторы, после чего опишем подробнее каждый из них:

- а) генератор случайного расписания;
- б) генератор случайного расписания, удовлетворяющего строгим ограничениям;
- в) генератор случайного расписания на каждый рабочий день, удовлетворяющего строгим ограничениям.

Генератор случайного расписания. Данный генератор итеративно генерирует смены от начала планируемого периода до конца так, что длина каждой смены и отдых между ними следует нормальному распределению. Несмотря на то, что случайное решение может не удовлетворять многим ограничениям, оно может довольно сильно помочь алгоритму в целом. У этого генератора можно выделить следующие особенности:

- а) для полученного решения не надо проверять строгие ограничения, что экономит процессорное время;
- б) тем не менее размер пространства решений, в котором алгоритму необходимо выбрать расписание, на несколько порядков больше, чем у других генераторов, из-за чего генерация может потратить даже больше процессорного времени;
- в) необходимо правильно подобрать параметры распределений для корректной работы генератора;

- г) случайное решение может иметь удачные смены в расписании, что при скрещивании с другим решением может привести к ранее неизведанному оптимуму;
- д) случайное решение может оказаться достаточно удачным, чтобы мутациями прийти к неожиданному неинтуитивному оптимуму (который не приближается другими генераторами), то есть увеличивается разнообразие популяции;
- е) тем не менее, решение может довольно сильно нарушать многие ограничения, в том числе и строгие, из-за чего такое решение имеет большую вероятность не пережить этап селекции эволюционного алгоритма.

Генератор случайного расписания, удовлетворяющего строгим ограничениям. У случайного расписания есть очевидный минус — оно с большой вероятностью нарушает строгие ограничения. Эту проблему можно решить, сузив пространство поиска решений с помощью строгих ограничений из постановки задачи. Расписание будет генерироваться итеративно: алгоритм будет поддерживать множество смен R , добавляя новую смену в него на каждой итерации. Будем считать, что S_i^+ — множество смен, удовлетворяющих строгим ограничениям на итерации i , а S_0^+ — множество всех смен. Данный алгоритм основывается на предположении $S_i^+ \subset S_{i+1}^+$, что верно при отсутствии ограничения на норму работы для каждого оператора. Для данного генератора будем считать, что ограничение на норму поставлено, как «выработка не больше нормы», а не «выработка равна норме». Опишем итерацию i данного генератора расписаний следующими этапами:

- а) построить множества S_i^+ с помощью фильтрации S_{i-1}^+ ;
- б) выбрать случайной смены $s_i \in S_i^+$;
- в) добавить смены s_i в расписание.

Алгоритм останавливается, когда $S_i^+ = \emptyset$. Очевидно, алгоритм остановится, так как S_1^+ конечно, и при фильтрации из S_{i-1}^+ удаляется хотя бы смена s_i . Фильтрацию множества S_0^+ до S_1^+ проводить бессмысленно, так как множество S_0^+ может быть бесконечно. Поэтому при реализации следует опираться на ограничения задачи. Например, можно за S_0^+ принять множество смен, удовлетворяющих ограничениям на длину и начало смены. Рассмотрим особенности данного генератора:

- а) эволюционный алгоритм всегда будет возвращать допустимый ответ, так как сгенерированные расписания уже допустимы — это даёт возможность проверить существование решения и ограничить время работы алгоритма;
- б) для каждого «хорошего» расписания есть ненулевая вероятность быть сгенерированным;
- в) тем не менее, сгенерированное расписание все еще плохо удовлетворяет мягким ограничениям.

Генератор случайного расписания на каждый рабочий день, удовлетворяющего строгим ограничениям. Решим основную проблему прошлого генератора — решение плохо удовлетворяет мягким ограничениям. Генератор будет оптимизировать ограничение на равномерность длин смен и ограничение на смену в каждый рабочий день. Пусть $S_{0,k}^+$ — множество смен длины m_k , где m_k — минимальная длина смены оператора k . Применим предыдущий алгоритм к $S_0^+ = \bigcup_{k=1}^K S_{0,k}^+$, получив при этом расписание T_0^+ . Далее рассмотрим шаги на итерации j данного алгоритма:

- а) выбрать случайную смену $t_j \in T_{j-1}^+$, которая еще не выбиралась на данной итерации;
- б) добавить шаг в конец смены t_j , получив при этом t_j^+ ;
- в) положить $T_j^+ = T_{j-1}^+ \setminus \{t_j\} \cup \{t_j^+\}$;
- г) если T_j^+ удовлетворяет ограничениям, перейти к следующей итерации, иначе вернуться к шагу (а).

Алгоритм останавливается, когда на первом шаге нет таких $t_j \in T_{j-1}^+$, что еще не выбирались на итерации j . Данный алгоритм остановится, так как T_0^+ конечно, и каждую итерацию суммарное число шагов в расписании увеличивается, что ограничено нормой. Рассмотрим особенности данного генератора:

- а) в отличие от прошлого алгоритма, высока вероятность, что в каждый рабочий день будет смена, так как ограничение на норму слабо влияет на выбор моментов начала смен;
- б) длина каждой смены имеет биномиальное распределение, которое сосредоточено около среднего значения, то есть длины смен будут примерно равны;
- в) тем не менее, некоторые смены, удовлетворяющие строгим ограничениям, не могут быть сгенерированы данным генератором.

В конечной версии алгоритма планирования расписания был выбран данный алгоритм, так как он лучше всего приближает решение к локальному оптимуму.

2.2.2. Генераторы перерывов

Генераторы смен были спроектированы так, чтобы логика не опиралась на ограничения. Множество смен фильтровалось предикатом «расписание удовлетворяет ограничением», а использование ограничений было неявным. Такое решение было принято, так как число ограничений на смены достаточно велико и сами ограничения склонны меняться со временем.

Перерывы же имеют не так много ограничений. Более того, перерывы должны генерироваться для каждой смены, то есть их генерация — критичное по производительности место. Поэтому было принято решение добавить в генератор «знание» об ограничениях.

Так как ограничения на перерывы не зависят от того, каким образом спланированы смены, перерывы можно генерировать независимо от начала смены. Заметим, что независимо от того, как перерывы будут сгенерированы, суммарная их длина будет одинаковой. Аналогично, одинаковой будет и суммарная длина всех рабочих периодов. Также одинаковым будет и число всех рабочих периодов. Тогда задачу генерации можно разделить на два независимых этапа:

- а) генерация типа для каждого перерыва;
- б) генерация длин рабочих периодов.

Этап (а) реализуется тривиально:

- а) создать список L длины N , где N — число перерывов, для которых необходимо сгенерировать тип;
- б) заполнить L типами перерывов — число повторений каждого типа перерывов задается ограничениями;
- в) генерируем случайную перестановку списка L ;
- г) назначаем тип перерывам, в порядке появления типов в L .

Причем первый перерыв не участвует в этом этапе, так как для него тип уже предопределен ограничениями.

Рассмотрим этап (б). Заметим, что ограничение на минимальную длину рабочего периода не вносит содержательных изменений в алгоритм. Если m^l , l и c — минимальная длина рабочего периода, суммарная длина рабочих пе-

риодов и число рабочих периодов соответственно, то достаточно применить алгоритм для $m^{l'} = 0$ и $l' = l - m^l \cdot c$, после чего к каждому полученному рабочему периоду прибавить m^l . Перечислим рассмотренные генераторы, после чего опишем подробнее каждый из них:

- а) почти биномиальный генератор;
- б) почти равномерный генератор;
- в) равномерный генератор.

Почти биномиальный генератор. При генерации смен уже обсуждался похожий алгоритм. Данный алгоритм итеративно добавляет шаг к случайному периоду, к которому ограничения позволяют добавить шаг. Таким образом, получается почти биномиальное распределение: вероятность генерации рабочего периода максимальной длины больше, чем в классическом биномиальном распределении, так как ограничения не позволяют сгенерировать длину больше максимальной.

Данный алгоритм прост в понимании, но наиболее вероятно сгенерирует равномерные по длине рабочие периоды. Несмотря на то, что данное свойство может быть полезным, постановка задачи не предполагает оптимизацию каких-либо мягких ограничений для перерывов. Поэтому далее будут рассматриваться другие генераторы.

Почти равномерный генератор. Рассмотрим такой алгоритм генерации рабочих периодов, который не учитывает ограничения на длину. На самом деле решаемая задача — генерация случайного сочетания с повторениями. Так же, как и для прошлого алгоритма, появление ограничений увеличит вероятность граничных случаев, из-за чего нельзя считать, что данный алгоритм равномерно генерирует рабочие периоды. Аналогично, из-за «нечестной» равномерности распределения далее рассмотрен более «честный» генератор.

Равномерный генератор. Равномерный алгоритм будет простым:

- а) сгенерируем все возможные варианты рабочих периодов для каждого набора входных параметров;
- б) вернем случайный список рабочих периодов из полученных вариантов.

Заметим, что на практике таких вариантов не так много. Например, для шага длиной в 15 минут, с максимальной длиной рабочего периода в два часа и не более, чем с пятью перерывами, потребуется хранить не более четырех миллионов чисел. Поэтому такой вариант не только оптимален, но и «честен»

по сравнению с остальными рассмотренными. Тем не менее, у такого алгоритма есть очевидный минус: при изменении параметров ограничений перерывов алгоритм может начать сильно тормозить и заполнять оперативную память. Поэтому для такого алгоритма нужна верхняя оценка по параметрам, с которыми алгоритм может работать.

2.2.3. Операторы мутации

Для оптимизации решения недостаточно реализовать генераторы, которые хорошо приближают локальный минимум. Более того, заметим, что генераторы не учитывают покрытие прогноза. Поэтому необходимо также реализовать операторы мутации, которые будут постепенно улучшать существующее решение. Были реализованы следующие операторы мутации:

- а) «добавление смены»;
- б) «починка нормы»;
- в) «равномеризация смен»;
- г) «сдвиг смены»;
- д) «сдвиг перерыва».

«Добавление смены». Это достаточно глобальная мутация: она может сильно испортить решение, и быстро починить его можно, только удалив эту смену. Поэтому полученное решение не должно ухудшать степень удовлетворения строгим ограничениям. Реализуем данную мутацию следующим образом:

- а) случайно определить оператора, у которого будет изменено расписание;
- б) создать смену минимального размера для каждого разрешенного начала смены;
- в) сгенерировать перерывы для каждой смены;
- г) удалить из полученного множества смены, добавление которых нарушает строгие ограничения;
- д) добавить случайную смену из полученного множества в расписание.

«Починка нормы». Перейдем к более локальным операторам мутации. Сгенерированные смены могут не удовлетворять ограничению нормы, поэтому нужна мутация, которая будет постепенно «чинить» это. Чтобы увеличить вероятность «выживания» расписания при эволюции, имеет смысл изменять длину граничной смены (смены минимальной/максимальной длины). Это поз-

волит уменьшить штраф за неравномерные смены при мутации. Подробнее, алгоритм будет следующим:

- а) случайно определить оператора, у которого будет изменено расписание;
- б) определить необходимое изменение количества работы за период: если норма не достигается, то количество работы необходимо увеличить, иначе уменьшить;
- в) случайно определить один из двух способов выбора смены:
 - 1) выбрать случайную смену;
 - 2) выбрать граничную смену (случайная смена минимальной длины, если количество работы необходимо увеличить, иначе случайная смена максимальной длины).
- г) выбрать смену способом из прошлого пункта;
- д) изменить размер смены исходя из пункта (б).

«Равномеризация смен». Оператор мутации должен приближать решение к минимуму. Перечисленные ранее операторы мутации не учитывают штраф на неравномерную длину смен, поэтому необходимо реализовать алгоритм, оптимизирующий это ограничение. Отсюда очевидно действие, которое будет совершать оператор: переносить шаг из одной смены в другую так, чтобы длины стали более равномерны. Смены могут выбираться случайно, тогда шаг будет переноситься со смены большей длины на смену меньшей длины. Но для ускорения сходимости имеет смысл переносить шаг со смены максимальной длины на смену минимальной длины. Подробнее, алгоритм будет следующим:

- а) случайно определить оператора, у которого будет изменено расписание;
- б) случайно определить один из двух способов выбора пары смен:
 - 1) выбрать случайную пару смен;
 - 2) выбрать случайную пару смен, первая из которых будет иметь минимальную длину, а вторая — максимальную.
- в) удалить последний шаг из смены большей длины и добавить в смену меньшей длины.

«Сдвиг смены». Операторы мутации, описанные ранее, плохо оптимизируют равномерность покрытия прогноза. Они оптимизируют мягкие или строгие ограничения. Основная задача планирования — расставить смены в правильной перестановке, потому что длина смены вряд ли будет отличаться

в оптимальных решениях. Поэтому предлагается добавить оператор движения смены. Причем заметим, что двигать смены маленькими шагами, может быть довольно трудозатратно, так как часто шаг не является допустимым началом смены. С другой стороны, движение маленькими шагами будет проходить плавнее, без «скачков» функции приспособленности. В целях ускорения алгоритма, было принято решение использовать первый вариант. Подробнее, алгоритм будет следующим:

- а) случайно определить оператора, у которого будет изменено расписание;
- б) случайно определить направление сдвига: смена будет двигаться в положительном или отрицательном направлении оси времени;
- в) найти следующее допустимое начало смены в выбранном направлении;
- г) переместить смену в направлении сдвига.

«Сдвиг перерыва». Аналогичный оператор сдвига необходим и для перерывов. Необходим он по аналогичным причинам: правильное расположение перерывов поможет покрыть прогноз более равномерно. Алгоритм будет следующим:

- а) случайно определить смену, у которой будет изменено расположение перерывов;
- б) случайно определить направление сдвига: положительное или отрицательное;
- в) переместить перерыв в направлении сдвига.

Выводы по главе 2

В данной главе был предложен алгоритм планирования смен операторов с учетом всех поставленных ограничений. Был разработан метод оценки алгоритма: неформальные требования были превращены в метрику, которую можно использовать для оценки качества решения. Были разработаны как генераторы, создающие стартовое решение, так и операторы мутации, которые постепенно улучшают стартовое решение, двигая его к локальному минимуму. Полученный алгоритм можно использовать для планирования работы команд служб поддержки.

Тем не менее, не были разработаны операторы скрещивания расписаний. Их сложнее оптимизировать, а также сложно интуитивно понять, как они будут улучшать решение. Поэтому исследование в этой области не проводилось.

ГЛАВА 3. РЕАЛИЗАЦИЯ И АПРОБАЦИЯ АЛГОРИТМА ПЛАНИРОВАНИЯ

3.1. Архитектура программного кода планирования смен

3.1.1. Программная реализация эволюционного алгоритма

Для проведения экспериментов был разработан программный код, использующий эволюционный алгоритм. Было принято решение спроектировать алгоритм в виде библиотеки. Такой код проще использовать: можно внедрить его как монолит, а можно — как отдельный микросервис. Более того, такой код можно будет использовать в одновременно нескольких сервисах и в разных вариантах.

Базовый функционал, сам эволюционный алгоритм, выделен в отдельный класс, который не зависит от конкретной задачи планирования. Это позволяет переиспользовать данную логику для решения других задач с помощью эволюционных алгоритмов. Более того, это позволяет довольно просто менять базовый алгоритм, не вникая в суть конкретной задачи. Также это упрощает код — абстрагирует части друг от друга, делает их независимыми.

Для реализации базового функционала использована программная платформа jMetal [13]. Для интеграции был использован шаблон проектирования «Мост», который позволит поменять программную платформу, если это требуется.

Каждая сложная, хорошо спроектированная система разбита на слои. Это упрощает дальнейшие изменения в слоях и чтение существующего кода. Такой принцип и принимался во внимание при проектировании модельных классов. Диаграмма зависимостей разработанных компонентов представлена на рисунке 1.

Модельный слой абстрагирует понятие времени. За единицу времени взят шаг, длительность которого определяется пользователем библиотеки. Все дальнейшие расчеты проводятся в шагах и в коде нет понятия «частичный шаг». Рассмотрим плюсы данного подхода:

- а) ошибки неправильного округления временных интервалов к шагам вынесены на уровень пользовательского кода;
- б) упрощение кода, перевод из одной модели в другую лежит в одном месте;
- в) относительная система координат — коду не нужно знать о том, с какого момента планируется расписание.

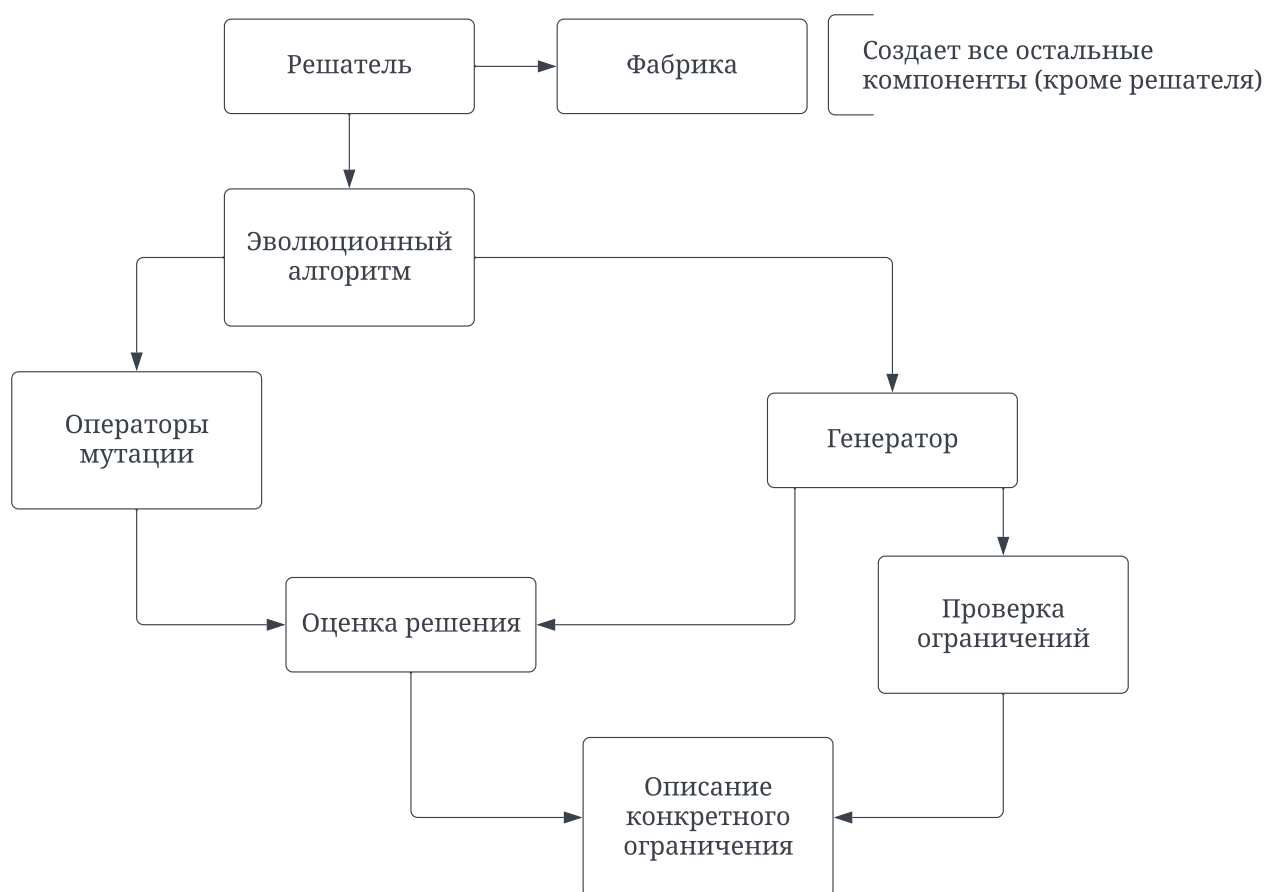


Рисунок 1 – Диаграмма зависимостей компонентов

Также в таком подходе есть и минусы:

- а) нельзя определить частичный шаг — для более точного расписания необходимо уменьшать длину шага, увеличивая размер задачи, даже если маленький шаг полезен не на всем периоде планирования;
- б) плохая репрезентативность моделей — по шагам трудно понять моменты начала и конца смены в календаре.

Также на уровне моделей объединяются некоторые ограничения, что позволяет задавать более абстрактные свойства операторам. Например, заданные начала смен для оператора объединяются с запретом на начало смены ночью или в выходной. Аналогично, для оператора задаются «запрещенные» шаги — шаги, с которыми смены не должны пересекаться. Так можно задать оператору не только отсутствие, но и, например, запретить работать ночью.

Дополнительно на модельном уровне абстрагировано понятие часового пояса, в котором работает оператор. Аналогично, данное решение было принято для упрощения кода и избежания ошибок, хотя и результаты становятся менее репрезентативными.

На модельном уровне также задаются данные для ограничений на перерывы. Например, допустимые длины рабочих периодов или число перерывов для каждой длины смены. Также есть возможность передать пустые ограничения — так в любом месте можно узнать, есть ли ограничения на перерывы.

Далее рассмотрим логику оценки расписания. Изначально было принято решение считать штраф в одном классе, так как существующее решение следовало такому же принципу. Позже было обнаружено, что старые ограничения часто изменяются, а также добавляются новые. Более того, такой подход нарушает принцип единственной ответственности и ухудшается понятность кода.

В итоге, было принято решение выделить пересчет штрафа по каждому ограничению в отдельный класс. Для подсчета штрафа в метод передается только расписание одного оператора, так как штрафы локальны по каждому оператору. Аналогичное замечание можно сделать и про штрафы за нарушение ограничений перерывов — такие штрафы локальны для каждой смены и их можно считать отдельно (и даже параллельно). Дополнительно, умножение штрафа на весовой коэффициент также было вынесено в отдельный класс. Такое решение позволило разделить ответственность, класс ограничения ничего «не знает» про весовой коэффициент. Тем не менее, пересчет штрафа за покрытие прогноза было решено оставить в старом классе, так как эти вычисления нельзя выполнить локально для каждой смены.

Так как генерация решения происходит итеративно необходим предикат, который проверяет, можно ли добавить смену в заданное множество смен. Данный предикат был построен аналогично подсчету штрафа — все предикаты были реализованы в одном классе. Предикаты были разделены по ограничениям и перенесены в тот же класс, в который перенесен пересчет штрафа. Рассмотрим отличительные особенности такого подхода:

- а) для прошлого решения при изменении ограничения необходимо изменить код в двух разных местах — предикате и пересчете штрафа (два разных файла);
- б) на замену огромному, трудно поддерживаемому классу получаем код, который разбит на несколько небольших частей;
- в) соблюдается принцип единственной ответственности;
- г) код проще, каждый класс содержит в себе только одну цель.

На основе предикатов, описанных выше, и построен генератор расписания. Логика разделена на две части: генерация смены и генерация перерывов. Такое разделение имеет смысл, так как ограничения на перерывы локальны для смены, то есть не зависят от момента начала смены.

В коде было принято решение улучшать расписание очень маленькими изменениями — так итерации будут исполняться довольно быстро и сам алгоритм можно будет ограничить по времени. Аналогичному принципу необходимо следовать и при генерации расписания. Для улучшения производительности кода добавим в генератор знание о допустимых началах смен и длинах смен — так генератор будет перебирать меньше возможных вариантов. В итоге, получим следующий алгоритм:

- а) перемешать допустимые начала смены случайным образом, получив список S^s ;
- б) в порядке появления в списке S^s , добавлять в расписание смену минимальной длины, если полученное расписание удовлетворяет ограничениям — получим список смен S ;
- в) пока суммарная длина смен меньше нормы, добавлять один шаг к случайной смене из S , для которой это позволяют сделать ограничения.

Аналогичному принципу быстрой генерации следует и генератор перерывов. Более того, к такому генератору чаще поступают запросы, так как перерывы должны генерироваться хотя бы для каждой смены, создающейся генератором смен. Для генерации перерывов возьмем уже описанный ранее равномерный генератор перерывов. В таком случае генерация перерывов будет иметь асимптотическую сложность $\mathcal{O}(1)$, что следует принципу быстрой генерации, описанному ранее. Также, чтобы оптимизировать использование памяти, будем вычислять каждый ответ лениво, как только по нему поступит запрос. Таким образом, для расписаний с небольшими сменами генератор будет занимать в несколько раз меньше памяти.

Сгенерированное решение необходимо постепенно приближать к оптимуму, для чего были разработаны операторы мутации. Были реализованы все перечисленные выше операторы мутации. Все модели заданы неизменяемыми, поэтому при мутации алгоритм вынужден создавать новый объект расписания с небольшими изменениями. У такого подхода есть следующие минусы:

- а) оперативная память довольно быстро заполняется предыдущими индивидуумами;
- б) операторы мутации тратят время на инициализацию новых объектов;
- в) усложняется код мутации: вместо небольшого изменения в изменяемом объекте необходимо копировать все расписание, указывая лишние данные, которые не изменились.

Тем не менее, такой подход имеет более значительные плюсы, из-за чего он и был выбран при реализации библиотеки:

- а) не нужно следить за тем, остался ли измененный объект в популяции;
- б) мутации потокобезопасны;
- в) в коде меньше ошибок.

На каждой итерации алгоритму необходимо выбрать, какой из операторов мутации использовать. Для этого необходимо выбрать один из операторов следуя какому-то распределению. Зададим такое распределение с помощью веса для каждого из операторов. Для операторов, которые двигают смены или перерывы, поставим вес больше, чем у остальных. В этом есть смысл, потому что это основные операторы, оптимизирующие покрытие прогноза.

Далее было принято решение добавить в библиотеку средства для инициализации алгоритма. С помощью шаблона проектирования «Фабрика» был реализован компонент, создающий компонент «Решатель», используя конфигурации, описанные выше.

«Решатель» реализует метод, принимающий максимальное число итераций и максимальное время исполнения. В этом методе компонент запускает алгоритм планирования, и он исполняется пока не начнет выполняться один из критериев останова.

Для хранения весов был выбран формат «.properties», который часто используется для хранения конфигурационных данных в программных комплексах, написанных на языке программирования Java. Для получения также был написана соответствующая логика. Для доступа к каждому из весов используется тип перечислений, поэтому в коде невозможно получить значение для несуществующего веса.

3.1.2. Ускорение эволюционного алгоритма

Одна из ключевых особенностей данного алгоритма — скорость планирования сильно зависит от скорости исполнения команд процессором. В це-



Рисунок 2 – «Граф пламени» алгоритма до оптимизации

лом, на типичном процессоре алгоритм работает довольно долго. При нескольких тысячах итераций исполнение может занимать несколько минут. Поэтому такой алгоритм стоит оптимизировать под современные архитектуры вычислительных машин.

Современные технологии не позволяют дешево ускорять одно ядро процессора бесконечно. Однако, современные архитектуры содержат несколько ядер, которые используются при параллельном исполнении в программе. Для того, чтобы такое решение принесло пользу, программа должна уметь распределять работу, синхронизировать потоки исполнения и агрегировать результаты. Все это ухудшает читаемость кода, увеличивает число ошибок и усложняет будущую поддержку. Тем не менее, алгоритм сильно ускоряется при применении такой методики, используя все ресурсы процессора.

Рассмотрим способы распараллелить построенный алгоритм. Тривиальным образом можно распараллелить мутацию и генерацию начальных решений. Более того, так как мутации довольно быстро работают, можно их группировать в одну задачу, чтобы сэкономить на получении задач из очереди. Тем не менее, такие оптимизации не реализованы в проекте, так как выбранная программная платформа не позволяет абстрактно задать такое поведение.

На рисунке 2 представлен «Граф пламени» при запуске алгоритма на реальной команде техподдержки. Фиолетовым цветом выделены вызовы, содержащие в себе подстроку «calculateFitness» — пересчет функции приспособленности. Отсюда видно, что как раз его и надо оптимизировать. Заметим, что пересчет штрафа — детерминированная функция. Для одного и того же расписания оператора всегда будет один и тот же штраф. Также заметим, что опера-

торы мутации меняют расписания только одного оператора, поэтому переиспользование прошлых штрафов для пересчета новых может сильно ускорить исполнение. Рассмотрим варианты переиспользования результатов прошлых подсчетов:

- а) сохранение результатов функции по каждому входным значениям с помощью шаблона проектирования «декоратор»;
- б) сохранение результатов функций в модели расписания путем добавления нового поля.

Рассмотрим отличительные особенности первого подхода:

- а) необходимо следить за потреблением памяти — удалять устаревшие данные либо по истечению времени, либо по переполнению используемой структуры данных;
- б) простейшая реализация такого подхода использует ассоциативный массив в виде хеш-таблицы — для эффективного использования такого подхода нужно уметь быстро сравнивать расписания, а такая задача довольно нетривиальна;
- в) тем не менее, такой подход очень удобен — для реализации шаблона «декоратор» можно использовать аспектно-ориентированное программирование и помечать оптимизируемые функции аннотациями.

Также рассмотрим отличительные особенности второго подхода:

- а) необходимо самостоятельно заполнять структуру данными, применить аспектно-ориентированный подход значительно сложнее;
- б) в неизменяемой модели расписания появляется изменяемый ассоциативный массив — необходимо решать проблемы синхронизации потоков;
- в) для модели нужно инициализировать структуру для каждого нового объекта — если объекты ранее встречались, старую структуру сложно переиспользовать;
- г) тем не менее, нет необходимости следить за утечками памяти — при сборке мусора удалятся неиспользуемые объекты расписаний, а с ними удалятся и сохраненные данные.

В конце концов, был выбран второй вариант, так как его использование влечет за собой менее ошибочный алгоритм.

Рассмотрим, как разработанный метод переиспользования вычислений можно применить к алгоритму планирования. Оптимизировать будем пересчет

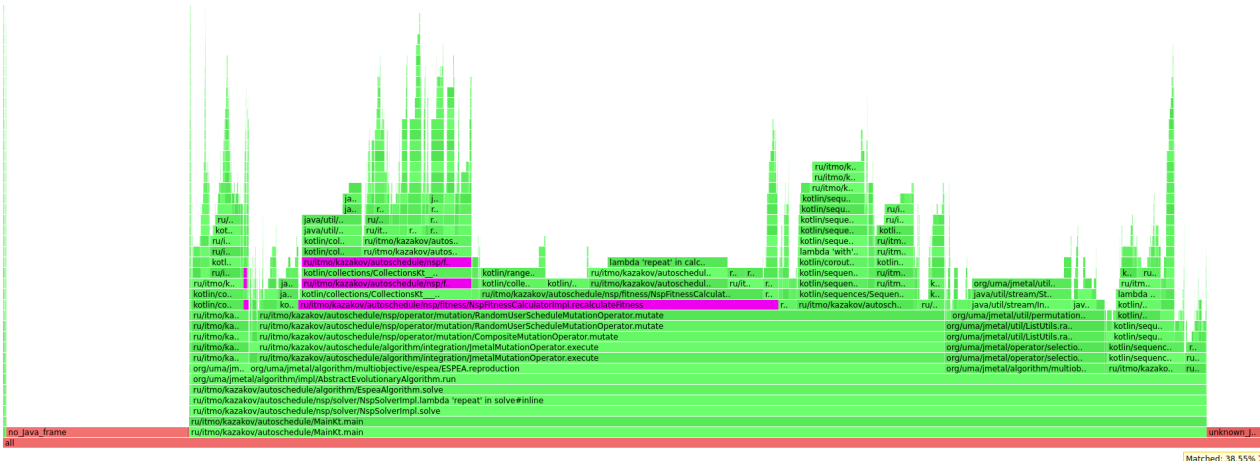


Рисунок 3 – «Граф пламени» алгоритма после оптимизации

штрафа, как уже было указано ранее. Реализуем логику, которая будет вычислять разницу штрафа по каждому оператору. Тогда для неизменившихся расписаний можно брать штраф из построенной ранее структуры. Для изменившихся смен будем использовать данные, посчитанные на прошлом шагу. Для этого метод, пересчитывающий штраф, будет принимать список добавленных смен, удаленных смен, новое и старое расписания. При использовании такого подхода, пересчет штрафа даже для большого расписания будет зависеть только от числа измененных смен.

Для штрафа за ограничения пересчет был оптимизирован тривиально: просто запустим подсчет штрафа по каждому оператору. Несмотря на кажущуюся неоптимальность, такой подход позволяет пересчитывать штраф только для операторов, у которых изменилось расписание. Дальнейшая оптимизация усложнит поддержку кода, поэтому не планируется ее проводить в ближайшее время.

Для пересчета штрафа покрытия прогноза на каждой итерации сохраняется покрытие прогноза по каждому шагу. Тогда можно тривиально добавить и удалить смены к покрытию и посчитать индекс дисперсии. Также можно оптимизировать подсчет самого индекса дисперсии, не используя те шаги, в которых число смен не изменилось. Такой подход, как и в прошлый раз, не был использован ради упрощения кода. «Граф пламени» после применения такой оптимизации представлен на рисунке 3. Как видно, процент времени, уходящий на пересчет функции приспособленности, уменьшился примерно в два раза.

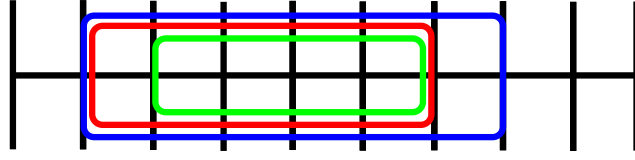


Рисунок 4 – Шаблоны смен в задаче планирования смен операторов

3.2. Сравнение алгоритмов планирования

3.2.1. Сравнение с решением задачи планирования дежурств медсестер

Для начала сведем задачу к задаче квадратичного программирования. Будем использовать способ аналогичный описанному в разделе 1.1.2. Более подробно, опишем, как можно определить шаблоны смен в задаче планирования дежурств операторов служб поддержки. В отличие от задачи планирования дежурств медсестер в шаблоны смен не заданы явно. Тем не менее, их можно определить следующим образом: будем считать, что $t_{s,l,r}^i$ — шаблон смены для i -го оператора длины l , который начинается с s -го допустимого начала смены оператора и имеет перерывы во время шагов из множества r . Пример шаблонов смен показан на рисунке 4.

Обозначим смену, соответствующую шаблону смены $t_{s,l,r}^i$, как $s_{s,l,r}^i$, а множество смен в расписании как S . Тогда можно задать двоичные переменные $a_{s,l,r}^i$, следующим образом:

$$a_{s,l,r}^i = \begin{cases} 1, & s_{s,l,r}^i \in S \\ 0, & \text{иначе} \end{cases}. \quad (36)$$

Далее, для каждого ограничения запишем неравенство для задачи квадратичного программирования:

- а) максимальная и минимальная длина смены: пусть $S^{b,1}$ — множество шаблонов смен с недопустимой длиной, тогда добавим ограничение

$$\forall i, s, l, r, t_{s,l,r}^i \in S^{b,1} : a_{s,l,r}^i = 0; \quad (37)$$

- б) дни начала всех смен одного оператора в его часовом поясе различны: пусть $D(s_{s,l,r}^i)$ — день начала смены $s_{s,l,r}^i$ в часовом поясе оператора i , а $S^{D,d,i} = \{(s, l, r) \mid D(s_{s,l,r}^i) = d\}$, тогда добавим ограничение

$$\forall d, i : \sum_{(s,l,r) \in S^{D,d,i}} a_{s,l,r}^i \leq 1; \quad (38)$$

- в) каждая смена должна пересекаться с горизонтом планирования: пусть $S^{b,2}$ — множество шаблонов смен не пересекающихся с горизонтом планирования, тогда добавим ограничение

$$\forall i, s, l, r, t_{s,l,r}^i \in S^{b,2} : a_{s,l,r}^i = 0; \quad (39)$$

- г) суммарная длина всех смен должна быть равна норме: пусть N_i — норма оператора i , тогда добавим ограничение

$$\forall i : \sum_{s,l,r} l a_{s,l,r}^i = N_i; \quad (40)$$

- д) смена оператора не должна пересекаться с отсутствием: пусть $S^{b,3}$ — множество шаблонов смен пересекающихся с отсутствиями, тогда добавим ограничение

$$\forall i, s, l, r, t_{s,l,r}^i \in S^{b,3} : a_{s,l,r}^i = 0; \quad (41)$$

- е) между сменами должен быть отдых (последовательные шаги, на которые оператор не назначен): пусть $R^{l,i}$ — минимальная длина отдыха между сменами в шагах для оператора i , а также $R_{s,l,r}^{n,i} = \{(s', l', r') \mid 0 \leq s' - s < l + R^{l,i}\}$, тогда добавим ограничение

$$\forall i, s, l, r : \sum_{(s', l', r') \in R_{s,l,r}^{n,i}} a_{s',l',r'}^i \leq 1; \quad (42)$$

- ж) смена оператора может начинаться только в заданные периоды начала смены: пусть $S^{b,4}$ — множество шаблонов смен, которые начинаются в недопустимые моменты, тогда добавим ограничение

$$\forall i, s, l, r, t_{s,l,r}^i \in S^{b,4} : a_{s,l,r}^i = 0; \quad (43)$$

- и) длины смен конкретного оператора не должны сильно различаться: пусть N_i и d_i — норма оператора i и число рабочих дней в планируемом периоде, тогда добавим ограничение

$$\forall i : \sum_{s,l,r} a_{s,l,r}^i \left(l - \frac{N_i}{d_i} \right)^2 \rightarrow \min. \quad (44)$$

Заметим, что ограничения (37), (39), (41) и (43) могут быть применены на этапе построения сведения. То есть алгоритм может использовать только те переменные $a_{s,l,r}^i$, для которых выполняются эти ограничения. Аналогичным способом зададим ограничения на перерывы. Если множество перерывов r не удовлетворяет ограничениям, то не будем добавлять переменную $a_{s,l,r}^i$.

Также заметим, что ограничение (44) нельзя тривиально добавить к ограничениям задачи линейного или квадратичного программирования. В таком случае, ограничение (44) необходимо добавить к оптимизируемой метрике. В качестве оптимизируемой метрики возьмем дисперсию покрытия прогноза, к которой прибавим покрытие прогноза: пусть p^i — производительность оператора i , а F_k — прогноз для k -го шага, $D(X)$ — несмещенная дисперсия списка X , тогда

$$A_k = \sum_{s \leq k < s+l, k \notin r} p^i a_{s,l,r}^i; \quad K_k = \frac{A_k}{F_k} \quad (45)$$

$$P^f = D(K) + \sum_k \frac{F_k - A_k}{F_k} \quad (46)$$

Так как $D(K)$ имеет в себе только квадратичные и линейные члены A_k , а $\sum_k \frac{F_k - A_k}{F_k}$ содержит в себе только линейные члены A_k , то такая функция подходит для оптимизации целочисленным квадратичным программированием. Сложив выражения (46) и (44), получим выражение, которое и будем оптимизировать целочисленным квадратичным программированием:

$$F = P^f + \sum_{i,s,l,r} a_{s,l,r}^i \left(l - \frac{N_i}{d_i} \right)^2 \quad (47)$$

К сожалению, такое сведение приводит к очень большому числу переменных $a_{s,l,r}^i$. На практике число шаблонов смен достигает порядка 1 000 000. Соответственно, матрица для такой задачи имеет размер порядка 1 000 000 x 1 000 000, что не вмещается в большинство современных устройств оперативной памяти. Более того, из-за ограничения (42) число неравенств в сведении достигает таких же порядков, что и число переменных.

Таким образом, такое решение немасштабируемо и требует очень дорогих ресурсов для поддержки. Поэтому такое решение не было выбрано в качестве основного.

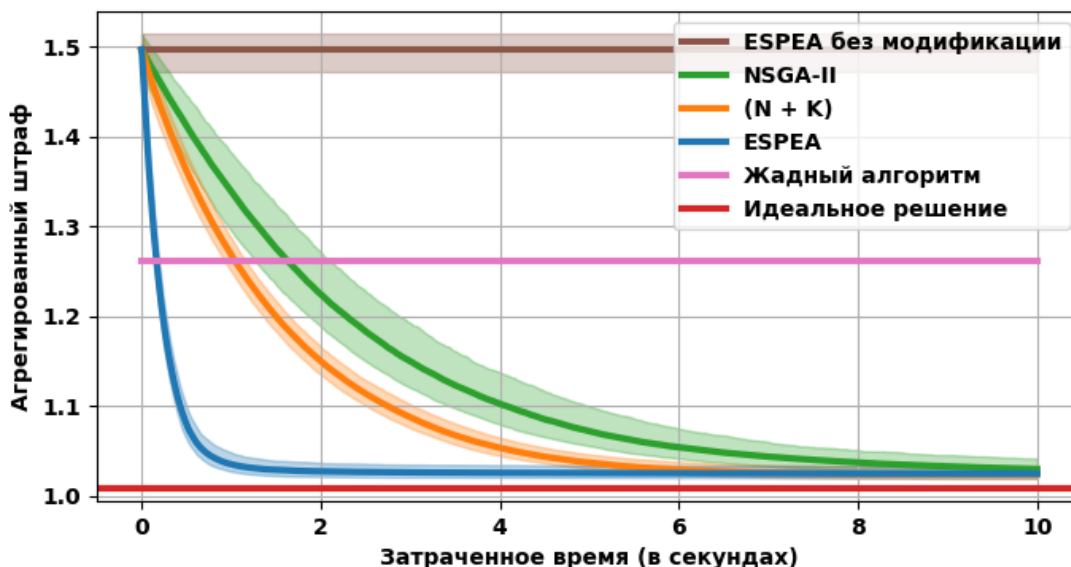


Рисунок 5 – Сходимость основных алгоритмов

3.2.2. Применение других эволюционных алгоритмов

Рассмотрим скорость сходимости разработанного алгоритма. Сходимость зависит как от операторов мутации, так и от выбранного базового эволюционного алгоритма. На рисунке 5 представлено сравнение рассмотренных алгоритмов в данной работе. По оси абсцисс указано время, за которое было получено решение с агрегированным штрафом, заданным по оси ординат. Заметим, что указанное время не учитывает время генерации начальных решений.

Замеры проводились на компьютере с процессором Intel(R) Core(TM) i7-11850H @ 2.50ГГц и 32ГБ оперативной памяти. Алгоритм запускался тысячу раз, принудительно останавливаясь через десять секунд после инициализации. Предварительно алгоритм запускался один раз, чтобы виртуальная машина применила оптимизации времени исполнения к алгоритму.

Для замеров была выбрана типичная команда операторов службы поддержки из порядка тридцати операторов. Расписание строилось на тридцать дней с шагом равным пятнадцати минутам. В качестве прогноза было выбрано покрытие какого-то решения. Более подробно, был взят прогноз из реальной службы поддержки и на нем было построено расписание с помощью алгоритма ESPEA. Затем полученное расписание было взято как идеальное решение, которое идеально покрывает прогноз, то есть прогноз равен покрытию дан-

ного расписания. Такой выбор прогноза позволяет оценить, насколько хорошо алгоритм приближает глобальный минимум.

Заметим, что выбранный алгоритм ESPEA показывает наилучшую сходимость среди всех рассмотренных алгоритмов. Тем не менее, на графике видно, что алгоритм NSGA-II медленнее сходится к минимуму, чем алгоритм $(N + K)$, хотя алгоритм NSGA-II также является многокритериальным. При подробном исследовании этого вопроса было выяснено, что алгоритм ESPEA за тот же период успевает исполнить в 102.14 ± 9.33 раз больше итераций, чем NSGA-II. Происходит это из-за того, что ESPEA основан на производительном быстром критерии выживания индивидуумов, чем NSGA-II. Более того, ESPEA игнорирует такие решения, которые доминируются какими-то существующими, что позволяет быстро отбросить решения, движущиеся в неправильном направлении.

Также на графике видно, что алгоритм $(N + K)$ сходится медленнее, чем ESPEA. Это происходит из-за того, что алгоритм скаляризует функцию приспособленности, теряя информацию о каждой компоненте функции приспособленности. Тем не менее данный алгоритм показывает неплохие результаты, имея сходимость лучше, чем у многих рассмотренных многокритериальных алгоритмов. Такие результаты получились из-за того, что задача имеет довольно мало критериев. Большинство ограничений выполняются для половины популяции, так как это гарантируется примененной модификацией. Поэтому остаются только два содержательных критерия — критерий покрытия прогноза и критерий равномерности длин смен. Второй критерий можно тривиально оптимизировать оператором мутации «равномеризация смен», поэтому при достаточно большом числе итераций алгоритм превращается в однокритериальный. Тем не менее, многокритериальный алгоритм позволяет развивать решение далее. Например, скорее всего многокритериальный алгоритм будет иметь лучшую сходимость, чем у $(N + K)$ при добавлении предпочтений операторов, как мягкое ограничение.

Также на графике представлено решение, основанное на алгоритме ESPEA и не использующее модификацию, описанную в разделе 2.2. Заметим, что решения почти не улучшаются. Такое поведение можно объяснить плохой выживаемостью хороших решений и малым разнообразием популяции. Если наивно оценивать каждое решение, то окажется, что решения с неболь-

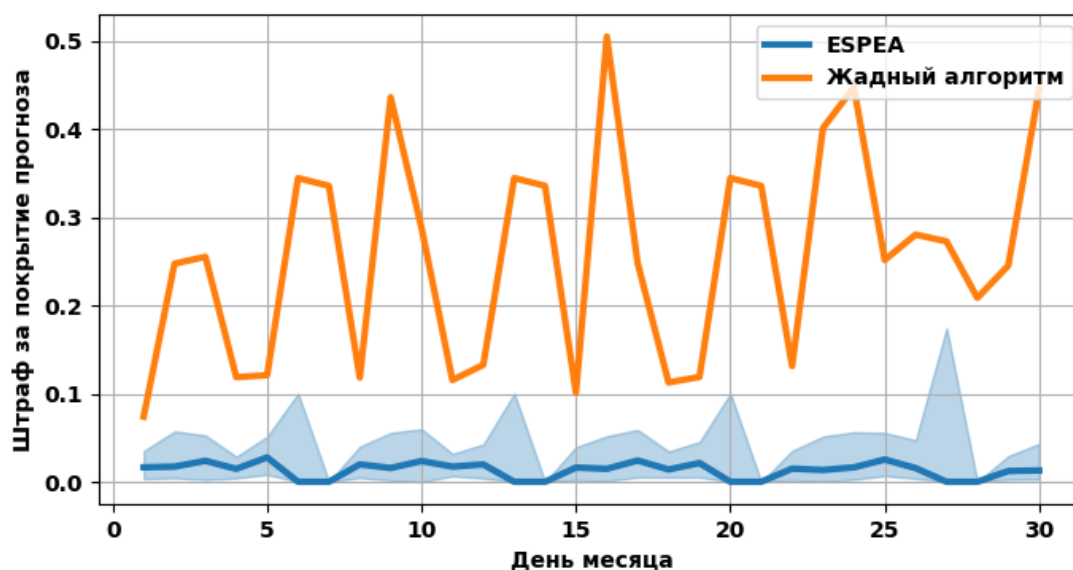


Рисунок 6 – Равномерность покрытия прогноза по каждому дню

шими нарушениями строгих ограничений довольно близки к решениям без таких нарушений. Из-за этого алгоритмы, которые поддерживают равномерное распределение решений, не смогут поддерживать разнообразие среди допустимых решений. Новые допустимые решения, скорее всего, будут удаляться, а старые будут неразнообразны.

Дополнительно, на графике представлено решение, полученное жадным алгоритмом. Жадный алгоритм не улучшает решение постепенно, поэтому на графике он представлен горизонтальной прямой. На рисунке 5 видно, что эволюционные алгоритмы дают результаты лучше, чем жадный алгоритм. На рисунке 6 представлена метрика покрытия прогноза, замерявшаяся независимо для каждого дня в планируемом периоде. Заметим, что эволюционный алгоритм статистически значимо лучше покрывает прогноз в каждый день.

Также заметим, что жадный алгоритм возвращает более оптимальное расписание, чем генераторы, используемые для эволюционных алгоритмов. Тем не менее, жадный алгоритм генерирует решение 100.66 ± 9.54 секунд, а предложенный генератор создает расписание за 4.97 ± 0.52 миллисекунд. Поэтому было принято решение взять быстрый генератор и исправить расписание эволюционным алгоритмом.

На рисунке 7 представлено сравнение дополнительно рассмотренных алгоритмов. Для построения этого графика использовались аналогичные пара-

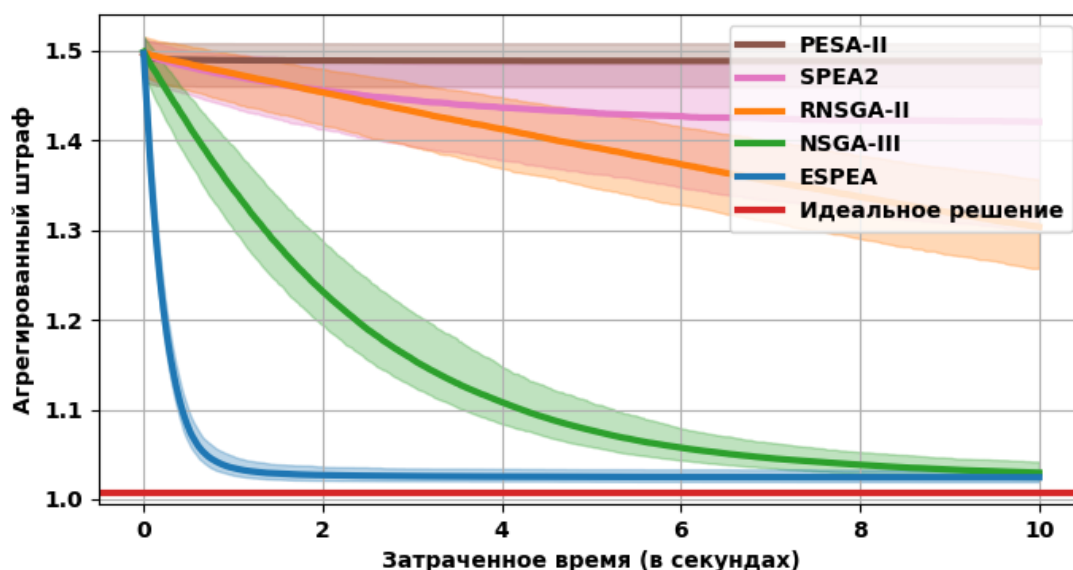


Рисунок 7 – Сходимость дополнительных алгоритмов

метры, как и для построения графика на рисунке 5. Также для сравнения на рисунок 7 был добавлен график сходимости алгоритма ESPEA.

Заметим, что качество решения для алгоритмов PESA-II и SPEA2 довольно быстро выходит на плато, но полученное решение сильно хуже, чем решение алгоритмом ESPEA. Более того, алгоритм PESA-II почти не улучшает решение — это связано с неудачным оператором выбора расписаний для мутации. В этой выборке расписания очень часто повторяются, из-за чего не все решения доходят до этапа мутации. Также на графике представлены алгоритмы RNSGA-II и NSGA-III. Эти алгоритмы также не показали скорость сходимости, как у алгоритма ESPEA. Как и в случае с NSGA-II, эти алгоритмы требуют много ресурсов процессора, из-за чего число итераций алгоритма ESPEA превосходит число итераций алгоритмов RNSGA-II и NSGA-III в несколько раз.

На рисунке 8 представлена сходимость алгоритма ESPEA при выборе различных генераторов.

Заметим, что генератор, равномерно распределяющий смены по дням, лучше приблизил начальное решение к минимуму, чем случайный генератор. Такое поведение наблюдается не только из-за того, что длины смен получаются равномерными, но и из-за того, что ресурс оператора более равномерно распределен по периоду планирования. Тем не менее, оба генератора доволь-

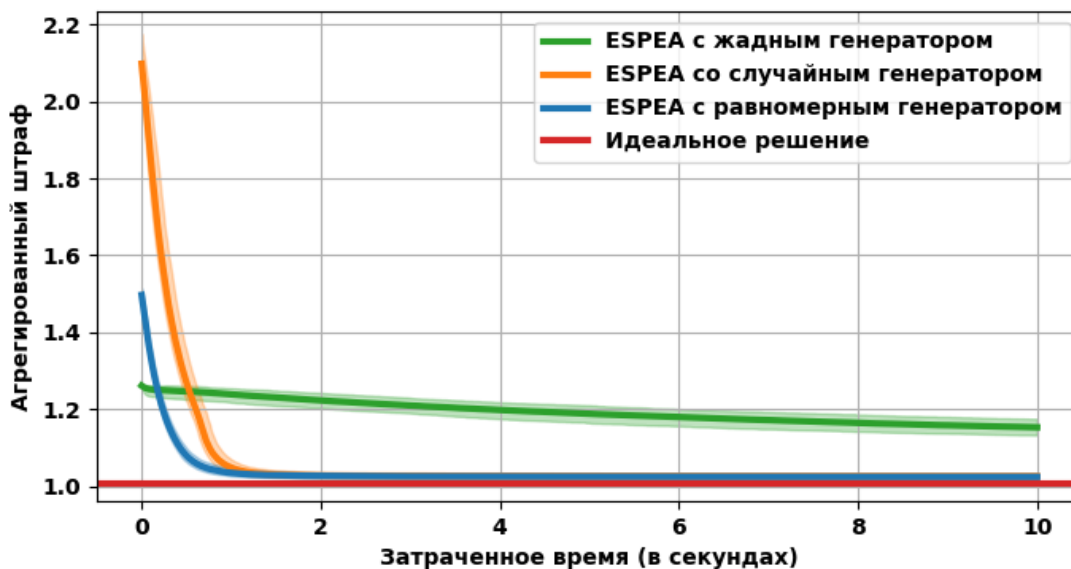


Рисунок 8 – Сходимость алгоритма ESPEA при выборе различных генераторов

но быстро приводят алгоритм к сходимости, то есть имеет смысл использовать оба.

Жадный алгоритм хоть и предложил лучшее решение, эволюционный алгоритм ESPEA не смог его улучшить так, как смог улучшить решения остальных генераторов. Такое поведение происходит из-за того, что жадный алгоритм генерирует расписания детерминированно, не используя генератор псевдослучайных чисел. Из-за этого в начальной популяции есть только одно расписание, что подразумевает отсутствие разнообразия в ней. А из-за отсутствия разнообразия алгоритм останавливается в одном локальном минимуме и не может найти альтернативные решения.

Выводы по главе 3

Таким образом, был разработан программный код, который реализует описанный алгоритм. Была описана архитектура приложения, а также рассмотрены возможные точки расширения. Экспериментально было показано, что выбранный алгоритм решает задачу лучше остальных рассмотренных. Было проведено сравнение с различными эволюционными алгоритмами, а также с жадным алгоритмом. Также было описано сведение к задаче целочисленного квадратичного программирования, которое на практике оказалось немасштабируемым.

В итоге, реализованный алгоритм был внедрен в компании Яндекс.

ЗАКЛЮЧЕНИЕ

В данной работе были рассмотрены ключевые особенности задачи планирования в службах поддержки. Была найдена аналогичная сфера деятельности, имеющая схожие особенности — планирование дежурств медсестер. Для задачи планирования в больницах были рассмотрены существующие решения, их преимущества и недостатки. Были рассмотрены как алгоритмы, позволяющие разделить задачу планирования на две задачи, так и алгоритмы, полностью решающие данную задачу.

Также была подробно описана задача планирования смен операторов служб поддержки. Были описаны все ограничения, поставленные в задаче, а также свойства, которые имеют операторы. Далее были рассмотрены различия задачи планирования медсестер и операторов, описаны причины возникновения трудностей при использовании существующих решений.

Для решения задачи планирования операторов службы поддержки было предложено решение. Была описана модификация классического эволюционного алгоритма, которая может улучшить сходимость алгоритма. Для каждого описания была задана как оценка вектором чисел, так и численная оценка, которая позволяет сравнивать любые два решения. Описаны генераторы начальных решений и операторы мутации, которые связали эволюционный алгоритм с решаемой задачей. Предложенные компоненты были сравнены и был выбран теоретически лучший вариант.

В итоге, предложенный алгоритм был реализован, как библиотека программного обеспечения. Была описана архитектура реализованной программы, рассмотрены точки развития. Также были описаны возможности масштабирования алгоритма, которые позволяют планировать расписание в современных системах.

Реализованный алгоритм был сравнен с похожими решениями. Было описано сведение к задаче квадратичного программирования, а также причины, почему оно немасштабируемо. Для сравнений использовались как разные генераторы, так и разные эволюционные алгоритмы. Также сравнение было проведено и с жадным алгоритмом. Экспериментально было показано, почему разработанный алгоритм показывает лучшие результаты.

Разработанный алгоритм имеет огромный потенциал роста. В проекте можно реализовать новые требования. Например, можно планировать распи-

сание для команд, в которых каждый оператор имеет множество навыков, а прогноз на каждый навык задан отдельно. Также довольно просто можно добавить в алгоритм учет личных пожеланий операторов.

Таким образом, поставленная задача была решена. Цель работы была достигнута, управление ресурсами служб поддержки было оптимизировано путем автоматизации планирования смен. В итоге, данная работа была использована при внедрении в компании Яндекс.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 A fast and elitist multiobjective genetic algorithm: NSGA-II / К. Deb [и др.] // IEEE Transactions on Evolutionary Computation. — 2002. — Т. 6, № 2. — С. 182–197. — DOI: 10.1109/4235.996017.
- 2 A greedy-based neighborhood search approach to a nurse rostering problem / F. Bellanti [и др.] // European Journal of Operational Research. — 2004. — Т. 153, № 1. — С. 28–40. — ISSN 0377-2217. — DOI: [https://doi.org/10.1016/S0377-2217\(03\)00096-1](https://doi.org/10.1016/S0377-2217(03)00096-1). — Timetabling and Rostering.
- 3 A multi-objective model for a nurse scheduling problem by emphasizing human factors / M. Hamid [и др.] // Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine. — 2020. — Т. 234, № 2. — С. 179–199. — DOI: 10.1177/0954411919889560.
- 4 *Aickelin U., Dowsland K. A.* An Indirect Genetic Algorithm for a Nurse Scheduling Problem // Computers & Operations Research. — 2004. — С. 761–778.
- 5 *Boyd S., Mattingley J.* Branch and bound methods // Notes for EE364b, Stanford University. — 2007. — Т. 2006. — С. 07.
- 6 *Braun M. A., Shukla P. K., Schmeck H.* Obtaining Optimal Pareto Front Approximations Using Scalarized Preference Information // Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. — Madrid, Spain : Association for Computing Machinery, 2015. — С. 631–638. — (GECCO '15). — ISBN 9781450334723. — DOI: 10.1145/2739480.2754674.
- 7 *Buzdalov M., Shalyto A.* A provably asymptotically fast version of the generalized Jensen algorithm for non-dominated sorting // Parallel Problem Solving from Nature–PPSN XIII: 13th International Conference, Ljubljana, Slovenia, September 13-17, 2014. Proceedings 13. — Springer. 2014. — С. 528–537.
- 8 *Cooper T. B., Kingston J. H.* The complexity of timetable construction problems // Practice and Theory of Automated Timetabling / под ред. Е. Burke, Р. Ross. — Berlin, Heidelberg : Springer Berlin Heidelberg, 1996. — С. 281–295. — ISBN 978-3-540-70682-3.

- 9 *Deb K., Jain H.* An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints // *IEEE Transactions on Evolutionary Computation*. — 2014. — T. 18, № 4. — С. 577–601. — DOI: 10.1109/TEVC.2013.2281535.
- 10 *Dowsland K. A., Thompson J. M.* Solving a nurse scheduling problem with knapsacks, networks and tabu search // *Journal of the Operational Research Society*. — 2000. — T. 51, № 7. — С. 825–833. — DOI: 10.1057/palgrave.jors.2600970.
- 11 *Droste S., Jansen T., Wegener I.* On the optimization of unimodal functions with the (1+1) evolutionary algorithm // *Parallel Problem Solving from Nature — PPSN V* / под ред. А. Е. Eiben [и др.]. — Berlin, Heidelberg : Springer Berlin Heidelberg, 1998. — С. 13–22. — ISBN 978-3-540-49672-4.
- 12 Fitness evaluation for nurse scheduling problems / E. Burke [и др.] // *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*. T. 2. — 2001. — 1139–1146 vol. 2. — DOI: 10.1109/CEC.2001.934319.
- 13 jMetal: A java framework for developing multi-objective optimization metaheuristics / J. J. Durillo [и др.] // *Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, ETSI Informática, Campus de Teatinos*, Tech. Rep. ITI-2006-10. — 2006.
- 14 *Kapelan Z. S., Savic D. A., Walters G. A.* Multiobjective design of water distribution systems under uncertainty // *Water resources research*. — 2005. — Т. 41, № 11.
- 15 PESA-II: Region-Based Selection in Evolutionary Multiobjective Optimization / D. W. Corne [и др.] // *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. — San Francisco, California : Morgan Kaufmann Publishers Inc., 2001. — С. 283–290. — (GECCO'01). — ISBN 1558607749.
- 16 *Widyastiti M., Aman A., Bakhtiar T.* Nurses Scheduling by Considering the Qualification using Integer Linear Programming // *TELKOMNIKA (Telecommunication Computing Electronics and Control)*. — 2016. — Сент.

- T. 14. — C. 933–940. — DOI: 10 . 12928 / TELKOMNIKA . v14i3 . 2913.
- 17 *Zitzler E., Laumanns M., Thiele L.* SPEA2: Improving the strength Pareto evolutionary algorithm // TIK-report. — 2001. — T. 103.