СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

ПО – программное обеспечение.

EAM – Enterprise Asset Management - система управления активами.

Фреймворк – программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

Makespan – период времени, который проходит от начала работы до ее завершения.

Job-shop – оптимизационная проблема в информатике и исследовании операций. Это вариант оптимального планирования заданий. В общей проблеме планирования заданий нам дается n заданий $J_1, J_2, ..., J_n$ с различным временем обработки, которые должны быть запланированы на m машинах с различной вычислительной мощностью, при этом мы пытаемся минимизировать время выполнения (makespan) - общую длину расписания.

SQL – structured query language - декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

.NET – это бесплатная, кроссплатформенная платформа для разработчиков с открытым исходным кодом для создания различных типов приложений.

ORM – Object-Relational Mapping - объектно-реляционное отображение, технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

ADO.NET – ActiveX Data Object для .NET - технология, предоставляющая доступ к данным и управление ими, хранящимися в базе данных или других источниках.

Entity Framework – ORM фреймворк с открытым исходным кодом для ADO.NET.

LINQ – Language Integrated Query - встроенный в .Net язык запросов к данным.

ВВЕДЕНИЕ

EAM-системы - чрезвычайно важная часть производственного процесса предприятий. Это инструмент управления производительностью, рисками и затратами, связанными с физическими (производственными) активами организаций.

Для современных производственных процессов актуальными являются задачи оптимизации: оптимизация маршрутов логистики, рабочего времени, производственных затрат и т. д. Масштаб таких оптимизационных задач может быть непосилен для решения сугубо человеческими ресурсами, даже при использовании EAM-систем, поэтому необходимо включать в область решения данных задач программно-аппаратные комплексы, решающие подобные задачи.

С 2011 года мировыми конгломератами ставится вопрос о четвертой промышленной революции - индустрии 4.0. Отходя от её утопических идей, индустрия 4.0 это гигантский шаг в сторону модернизации всех аспектов производства.

Большие данные, интернет вещей, блокчейн, искусственный интеллект, цифровые двойники - все эти технологии, которые уже нашли те или иные способы применения в различных областях науки и техники, только и ждут своего часа применительно к производству и производственным процессам. Уже сейчас самые передовые производства невероятно трудно представить без всех этих технологий.

Так новый передовой завод Intel [1] - полностью автоматизированное цифровое производство, где все самые важные и ответственные роли исполняют именно роботизированные механизмы или автономные дроны с интеллектом роя.

На текущий момент на рынке представлено достаточно много решений, связанных с EAM и ERP системами, как отечественные:

– 1С ERP

– ”TRIM”
– ”Галактика ERP”

Поменял местами трим и 1с

так и зарубежные:
– SAP ERP
– IBM Maximo

Все представленные EAM-системы (как отдельные, так и в составе ERP) позволяют в полуавтоматическом режиме планировать работы, то есть указать какое количество ресурсов и когда понадобится для обслуживания тех или иных активов.

Во всех приведённых выше системах данное планирование происходит по потребности, то есть в долгосрочной перспективе не учитывает внезапные изменения, которые могут произойти с ресурсами, и их всевозможные коллизии - ситуации, когда в один момент времени один и тот же ресурс может быть задействован в разных работах.

Исходя из этого можно сказать, что в таких системах стоит до сих пор нерешенная проблема актуализации ресурсного планирования. Именно решение этой проблемы является целью данной работы.

В качестве объекта работы выступают данные, полученные из системы класса EAM, структурированные определенным образом. В общем виде - база данных с набором сущностей вида ”работа”, ”тип работы, ”исполнитель” и так далее.

Цель данной работы - создание открытого мультиплатформенного решения, предназначенного для ресурсного планирования в системах управления физическими активами.

В данной работе решается задача ресурсного планирования, то есть распределение заданий по исполнителям в заданном временном интервале с учетом ограниченного количества имеющихся ресурсов.

В результате работы планируется получить готовое программно-техническое решение, которое может взаимодействовать с имеющимися

EAM-системами и обеспечивать для них решение задачи ресурсного планирования с заданными граничными условиями.

В процессе работы был проведен анализ доступных средств и методов в области планирования ресурсов, была составлена модель, отражающая в общем виде структуру ресурсов, для которой необходимо проводить процедуры планирования, а также проведен эксперимент, показывающий соответствие модели поставленной задаче.

Работа состоит из Введения, 3 глав и Заключения, содержит список литературных источников из 20 наименований и приложения.

1 Анализ предметной области разработки, сравнение с аналогами

1.1 Описание предметной области с анализом существующих решений и их недостатков исходя из целей работы и критериев сравнения, которые должны соответствовать искомым решениям поставленных задач.

Одной из областей в операционных исследованиях является составление и оптимизация расписаний. Когда количество работ, человеческих ресурсов и различных механизмов невелико, то составление расписания вполне посильная для человека задача. Но когда количество объектов, учитываемых при планировании, возрастает, например, до уровня предприятия, то задача перестаёт быть тривиальной даже для целого отдела планирования. В таких случаях становится необходимым использование средств автоматизации планирования, примером которых являются мультиагентные системы или программные комплексы комбинаторной оптимизации - отдельные или в рамках EAM-систем.

В целом можно разделить существующие методы построения и оптимизации подобных моделей на два основных подтипа - Мультиагентные системы (MAC) и Исследование операций (Operations research - далее OR) в рамках целочисленного программирования (программирования в ограничениях).

1.1.1 MAC

Мультиагентная система – система, состоящая из множества взаимодействующих друг с другом агентов. Каждый агент в такой системе независим и не имеет представления о системе в целом, но знает свои задачи и потребности, а также задачи и потребности агентов на одном с ним уровне. Для простоты понимания можно условно сказать, что агенты представляют собой

нечто схожее с NPC (от англ. Non-Player Character - Неигровой персонаж) в компьютерных играх.

Перевод с английского

Мультиагентные системы начали широко развиваться в девяностых годах и достигли своего пика к середине двухтысячных годов, после чего парадигма создания агентных сетей как чего-то, требующего отдельных подходов умерла в связи развитием и упрощением программирования в целом.

Согласно статьи Virginia Dignum и Frank Dignum [2], несмотря на существование целых организаций по изучению мультиагентных систем, ни за время существования этих организаций, ни за время их работы, MAC не смогли проявить себя ни в одной из заявленных областей, кроме как моделирования поведения персонажей в компьютерных играх [3]. Исследователи связывают неудачу MAC со слабой способностью конкурировать с другими компьютерными технологиями того времени, в первую очередь вызванной тем, что из-под пера ассоциации или заинтересованных исследователей так и не вышло платформы MAC, которую могли бы использовать все желающие за пределами «закрытого» сообщества. Второй проблемой при попытке применить исследования агентов на практике, по их мнению, стала неспособность охватить или учесть существенные характеристики проблемы, для решения которой используются агенты. Аналогичные выводы были представлены Городецким и Скобелевым в статье 2017 года [4].

Убрал название статьи

В целом всё-таки можно сказать, что MAC – достаточно перспективное направление, второе дыхание которому могут дать современные разработки в области искусственного интеллекта, нейросетей и программирования.

Главным плюсом (из которого в общих случаях следует и главный минус) мультиагентных систем является скорость реакции на изменения в системе и внешние факторы. Предполагается, что такая система фактически моментально должна реагировать на «происшествия» и сразу же выдавать удовлетворяющее решение с приемлемым качеством. Данное свойство муль-

тиагентных систем делает их максимально пригодными для использования в качестве систем решения задач реального времени или в качестве модели-двойника.

Под моделью-двойником представляется виртуальный клон реальной системы, где все объекты взаимодействия являются агентами. Предполагается, что такие модели будут использоваться, например, для симулирования внештатных ситуаций в системе и оценки реагирования системы и агентов на эти ситуации.

Так как МАС предлагают в целом «приемлемые решения», то в областях, где важна каждая мелочь и требуется наиболее полный точный результат, такой подход не подойдет. МАС вполне можно использовать на этапах конкретной эксплуатации, где как раз необходимы оперативные принятия решений, а вот в задаче установки оптимальных начальных условий такой подход скорее всего будет неприменим.

1.1.2 Исследование операций

Для получения наиболее точных и оптимальных решений применяется дисциплина под названием «Исследование операций». Согласно Википедии [5], «ИО - дисциплина, занимающаяся разработкой и применением методов нахождения оптимальных решений на основе математического моделирования, статистического моделирования и различных эвристических подходов в различных областях человеческой деятельности. Иногда используется название «математические методы исследования операций».

Методы исследования операций являются основными в решениях таких задач как планирование, транспортные потоки, маршрутизация, упаковка и задачи о назначениях. В варианте, представленном в статье, предлагается использовать методы комбинаторной оптимизации, а именно – целочисленное программирование в ограничениях.

Программирование в ограничениях — это теория решения комбинаторных задач, опирающаяся на большое количество знаний из областей искус-

ственного интеллекта, информатики и исследования операций. В программировании в ограничениях декларативно задаются ограничения для набора переменных входных данных. Ограничения не определяют последовательность действий для получения искомого результата, а указывают свойства и особенности для нахождения конечного результата.

Оптимизация с ограничениями, или программирование с ограничениями (англ.- constraint programming, далее CP), — это название, данное идентификации выполнимых решений из очень большого набора возможных решений. CP основывается на выполнимости (нахождении выполнимого решения), а не на оптимизации (нахождении оптимального решения) и фокусируется на ограничениях и переменных, а не на целевой функции. Фактически, задача CP может даже не иметь целевой функции - цель может заключаться просто в том, чтобы сузить большое множество возможных решений до более управляемого подмножества путем добавления ограничений к задаче.

1.2 Разработка возможных направлений проведения исследований и методов решений отдельных задач

Подкорректирвал название

Исходя из требований следует, что разработанная система будет предназначена для перепланирования некоторого множества предварительно заданных в EAM-системе работ.

На текущий момент основным средством ресурсного планирования является фактически ручной метод распределения работ, при котором ответственные люди вручную указывают потребность в работах и ресурсах. При больших объемах работ при таком типе планирования теряется одна из важных потребностей планирования, а именно потребность в актуальности расписания.

Когда количество работ велико, становится физически невозможно одними человеческими силами отследить или как-то исправить пересечения

в потребности тех или ресурсов, так как таких коллизий может быть сотни или тысячи даже на промежутке в неделю.

Решение данной проблемы программными средствами тоже сама по себе нетривиальная задача, так как в таком случае на систему необходимо смотреть в целом, а не разрешать отдельные коллизии, а каких-либо общепринятых адекватных решений для данной проблемы до сих пор представлено не было.

Исходя из вышесказанного одним из главных направлений проведения исследований в данной области будет анализ доступных средств и инструментов, подходящих под задачи ресурсного планировании, и вследствие исследование конкретных методов решения поставленных задач уже у выбранного направления.

1.3 Обоснование применяемых технологий и инструментов.

Реализация различных методов комбинаторной оптимизации представляется достаточно трудоемкой задачей, поэтому предполагается использование готовых библиотек и фреймворков, реализующих методы исследования операций.

Одними из самых популярных фреймворков для решения задач исследования операций являются открытые GLPK [6], LP_Solve [7], MiniZinc [8], Google OR-Tools [9], CHOCO [10] и проприетарные IBM CPLEX [11] и Gurobi [12].

К дальнейшему рассмотрению предлагается три из приведенных выше решений – Открытый и бесплатный Google OR-Tools, как целевой фреймворк в данной работе, и проприетарные IBM CPLEX и Gurobi, как лидирующие решения в отрасли.

Обоснование выбранных для сравнения фреймворков

Приведем частичные результаты бенчмарков между Google OR-Tools и IBM CPLEX, полученные в статье ”Google vs IBM: A Constraint Solving Challenge on the Job-Shop Scheduling Problem” [13], в таблице 1.1.

Таблица 1.1 – Сравнение Or-Tools и CPLEX в задаче Job-shop (меньше - лучше)

Номер теста	Одно ядро		Четыре ядра	
	CPLEX msp (sec)	OR-Tools msp (sec)	CPLEX msp (sec)	OR-Tools msp (sec)
1	1234 (1.9)	1234 (1.8)	1234 (3.3)	1234 (1.6)
2	943 (0.7)	943 (0.7)	943 (1.4)	943 (0.4)
3	656 (1169.3)	660	565 (525)	661 (1200)
4	682	679	680	679
5	685	695	694	689
6	55 (0)	55 (0)	55 (0)	55 (0)
7	930 (3.8)	930 (5)	930 (5.9)	930 (2.9)
8	1165 (1.4)	1165 (5)	1165 (0.5)	1165 (3.4)
9	666 (0)	666 (0.1)	666 (0)	666 (0.1)
10	655 (0.3)	655 (0.3)	655 (0.5)	655 (0.1)

В тестах производительности между Google OR-Tools и IBM CPLEX стоял вопрос решения проблемы составления расписания работы в цехе (англ. Job-shop scheduling problem), которая приобрела особую известность благодаря своей простой формулировке, приводящей к трудно решаемым задачам.

Наиболее типичным критерием оптимизации является минимизация промежутка времени (англ. makespan), т.е. промежутка времени между началом первой операции и окончанием последней. Проблема представляется в виде набора заданий, которые должны быть обработаны набором машин. Каждое задание - это последовательность операций, каждая операция должна быть обработана определенной машиной и занимает определенное время обработки. Каждое задание имеет определенный порядок операций, который должен соблюдаться. Допустимым решением этой задачи является такая последовательность операций на каждой машине, при которой нет перекрытия времени между двумя операциями на одной машине и соблюдается порядок операций.

Как видим из статьи, по результатам тестов на момент 2017 года фреймворк IBM CPLEX в целом чаще выигрывал у OR-Tools в размере makespan и быстроте получения решения, при этом фреймворк OR-Tools показывал себя лучше в многоядерных тестах большого масштаба.

В другой статье [14] представлены уже результаты сравнения коммерческих IBM CPLEX и Gurobi. По результатам тестов последние версии обоих решателей в целом идут нога в ногу и выдают результаты очень близкие как по качеству, так и по скорости нахождения решений.

По результатам бенчмарков можно сказать, что между фреймворками Google OR-Tools, IBM CPLEX и Gurobi (особенно в многоядерной конфигурации) существует условный паритет по скорости и качеству решения оптимизационных задач, и в целом предугадать, кто окажется впереди в условиях реальной задачи невозможно.

Одним из наиболее доступных фреймворков является Google Or-Tools. OR-Tools - это программное обеспечение с открытым исходным кодом для комбинаторной оптимизации, которая направлена на поиск наилучшего решения проблемы из очень большого набора возможных решений. Последние несколько лет именно OR-Tools среди всех открытых библиотек занимает первые места в соревнованиях MiniZinc Challenge по программированию в ограничениях [15]. Также, в отличие от большинства остальных фреймворков имеет большее количество официально написанных интерфейсов под разные языки программирования – C++ (изначально написан именно на нём), C#, Java, Python.

Фреймворк Google OR-Tools поддерживает решатели под задачи линейной оптимизации (решения проблемы, смоделированной как набор линейных зависимостей.), целочисленной оптимизации (Смешанное целочисленное программирование - Mixed Integer Programming – MIP; некоторые переменные обязательно представляют собой целые числа) и программирование в ограничениях. Все эти решатели могут быть использованы для решения таких задач, как: решение судоку, диета Стиглера [16], шахматные задачи, планирование, составление оптимальных маршрутов и т.д.

В результате обзора и анализа методов решения поставленной задачи, на данный момент наиболее подходящим методом можно признать метод комбинаторной оптимизации и программирования в ограничениях, так как он явно предоставляет набор устоявшихся практик и решений, показавших себя во множестве проектов.

Основным средством предлагается использовать фреймворк Google OR-Tools, решающий задачи комбинаторной оптимизации, так как данный фреймворк является бесплатным, открытым, может быть использован в коммерческой разработке и покрывает большую часть областей и задач исследования операций.

Мультиагентные системы на данный момент, напротив, по результатам анализа показали себя непригодными к повсеместному использованию, так как не предоставляют доступных средств для реализации и не были широко протестированы в реальных условиях.

1.4 Уточненная постановка задачи и требования к прототипу решения.

Имеются следующие начальные условия:

- Есть множество запланированных работ (ЗР) на заданном временном интервале. Каждая работа имеет расчетные плановую дату/время начала выполнения и плановую дату/время завершения выполнения (разница между этими датами определяет длительность работы).
- Каждая работа имеет перечень исполнителей, необходимых для ее выполнения. Для каждого исполнителя задана потребность, выраженная в чел.*час. В качестве исполнителя может быть указан один из конкретных исполнителей, имеющихся в штатном расписании.
- Имеется подразделение (цех, участок и т.п.), располагающее определенными людскими ресурсами. Состав людских ресурсов определяется штатным расписанием.

Задача:

На основании имеющихся исходных данных постараться построить такой порядок выполнения работ, чтобы в каждый день суммарная потребность в людских ресурсах и механизмах на выполнение всех работ не превышала объем располагаемых ресурсов, заданных в штатном расписании.

Для того, чтобы найти такое решение, система может перемещать исходные ЗР во времени. При этом должны соблюдаться следующие условия:

– длительность работ и потребность в ресурсах, указанные для каждой ЗР, должны сохраняться;
– порядок выполнения работ одного типа и для одного и того же объекта должен сохраняться;
– может быть задана плановость определенных типов работ.

Результат:

В результате система должна предложить новые плановые даты выполнения каждой ЗР при условии соблюдения заданных ограничений. Если решения нет, то система должна сообщить об этом, желательно при этом указать, какого ресурса или механизма (по мнению системы) недостаточно для выполнения плана.

1.5 Выводы

Были проанализированы два подхода к решению проблемы планирования в системах управления активами. Так как в самых популярных на сегодняшний день EAM-системах, таких как 1С ERP, ”ТРИМ”, зарубежных SAP и Maximo есть лишь стандартные полуавтоматические способы планирования работ, то одним из самых востребованных направлений по улучшению такого функционала является полностью автоматическое планирование по заданным потребностям с учетом актуального распределения ресурсов.

Двумя самыми подходящими для такого функционала подходами являются мультиагентные системы и программирование в ограничениях. По результатам анализа подход с применением мультиагентных систем сложно рассматривать в качестве основного для данного типа задач, так как до сих

пор, в том виде в каком он был представлен изначально, не показал реальных практических результатов, а также не было представлено каких-либо общедоступных сред для мультиагентного программирования. Программирование в ограничениях, напротив, предоставляет наиболее понятный набор инструментов и практик, поэтому для решения проблемы ресурсного планирования был выбран именно такой подход.

2 Проектирование

Проектирование планировщика ресурсов предполагает собой разработку, состоящую из следующих этапов:

– анализ доступных средств платформы .Net;
– выбор типа базы данных и её провайдера;
– определение алгоритмов, по которым будет работать программный модуль;
– написание и отладка программного модуля.

2.1 Проектирование архитектуры прототипа системы решения, включая разработку структуры (компоненты и связи между ними), внутреннего и внешнего функционирования.

2.1.1 Анализ доступных средств платформы .Net

На данный момент платформу .Net можно условно поделить на две части - .Net Framework и .Net Core.

На данный момент платформа .Net Framework является поддерживаемой, но устаревшей, и по рекомендациям Microsoft [17] не следует начинать новые проекты на этой платформе. Дополнительно необходимо указать, что платформа .Net Framework не является кроссплатформенной и работает только в операционных системах семейства Windows, что существенно ограничивает возможности современной разработки на ее основе на данный момент.

Открытый кроссплатформенный фреймворк .Net Core является преемником .Net Framework. Именно Core версию Microsoft рекомендует для всех новых проектов. Главным отличием новой платформы в первую очередь стала поддержка работы в Unix-системах (Linux, MacOS), соответственно все сервисы, написанные на .Net Core, без каких-либо ограничений могут быть

использованы в современной серверной парадигме: Linux серверы, Docker, контейнеризация и виртуализация.

Для данного проекта была выбрана самая акутальная версия фреймворка .Net Core - .NET 6, так как она является LTS-версией, а так же соответствует всем требованиям проекта - открытости и кроссплатформенности.

В качестве платформы для взаимодействия с пользователем был выбран серверный вариант фреймворка Blazor. Blazor - это бесплатный и открытый фреймворк для написания веб приложений на языке C#, поддерживающий реактивность, то есть работающий напрямую с DOM-элементами, а не HTML страницами. Blazor делится на Blazor WebAssembly - полностью клиентское приложение и на Blazor Server - клиент-серверную реализацию, в которой пользовательская и серверная часть общаются между собой по протоколу SignalR. SignalR - это бесплатная библиотека с открытым исходным кодом для Microsoft ASP.NET, которая позволяет серверному коду отправлять асинхронные уведомления веб-приложениям на стороне клиента. Библиотека включает в себя серверные и клиентские компоненты JavaScript.

2.1.2 Определение алгоритмов, по которым будет работать программный модуль

За планирование и/или перепланирование в планировщике будет отвечать библиотека Google OR-Tools. Задача, изложенная выше в первой главе, однозначно подходит под определение задач целочисленного программирования в ограничениях, так как каждый объект виртуальной модели в данной задаче можно определить как целочисленную переменную или их множество, на которые в явном виде можно наложить ряд ограничений и целевых функций, соответствующих условиям задачи.

Для такого типа задач пакет Google OR-Tools предоставляет отдельный класс для решения задач целочисленного программирования в ограничениях - CP-SAT, где CP (от англ. Constraint Programming - Программирование в ограничениях) - непосредственно целочисленное программирование в ограниче-

ниях, а SAT (от англ. Satisfiability - Удовлетворимость) - указывает, что целью решения данного класса задач является не нахождение одного конкретного результата, а нахождение одного или нескольких решений, удовлетворяющих ограничениям, наложенным на входные данные.

2.1.3 Выбор типа базы данных и её провайдера.

В первую очередь при выборе типа баз данных встаёт вопрос об использовании реляционных или нереляционных баз данных. В целом в EAM-системах обычно используются именно реляционные базы данных, так как этому способствует необходимость в четкой иерархической структуре модели представления активов предприятия. Также в EAM-системах редко встречается необходимость в шардинге - то есть в горизонтальном расширении данных для обеспечения масштабируемости. Исходя из этого для данного проекта была выбрана именно реляционная база данных.

В качестве реляционной СУБД был выбран PostgreSQL. PostgreSQL - это бесплатная система управления реляционными базами данных с открытым исходным кодом, в которой особое внимание уделяется расширяемости и соответствию стандарту SQL.

В данный момент ввиду острой необходимости в импортозамещении и независимости от зарубежных ПО, сервисов и средств именно СУБД PostgreSQL видится лучшим вариантом. У российских разработчиков уже есть большой опыт в развитии своих доработок и ответвлений PostgreSQL. Так, например, давно и успешно применяется ответвление Postgres Pro [18], СУБД ЛИНТЕР-ВС на базе PostgreSQL используется в ОС МСВС, а в дистрибутив операционной системы Astra Linux Special Edition (версия для автоматизированных систем в защищённом исполнении, обрабатывающих информацию со степенью секретности «совершенно секретно» включительно) включена СУБД PostgreSQL, доработанная по требованиям безопасности информации. Особенно важно, что СУБД Postgres PRO входит в реестр российского ПО [19].

2.1.4 Проектирование архитектуры

Желаемая архитектура системы представлена на рисунке 1. Предполагается, что разрабатываемая система ресурсного планирования будет являться отдельным сервером планирования, работающим независимо от EAM-системы. Связь между EAM-системой и сервером планирования должна осуществляться посредством наличия общей базы данных и API для передачи команд и промежуточных объектов.

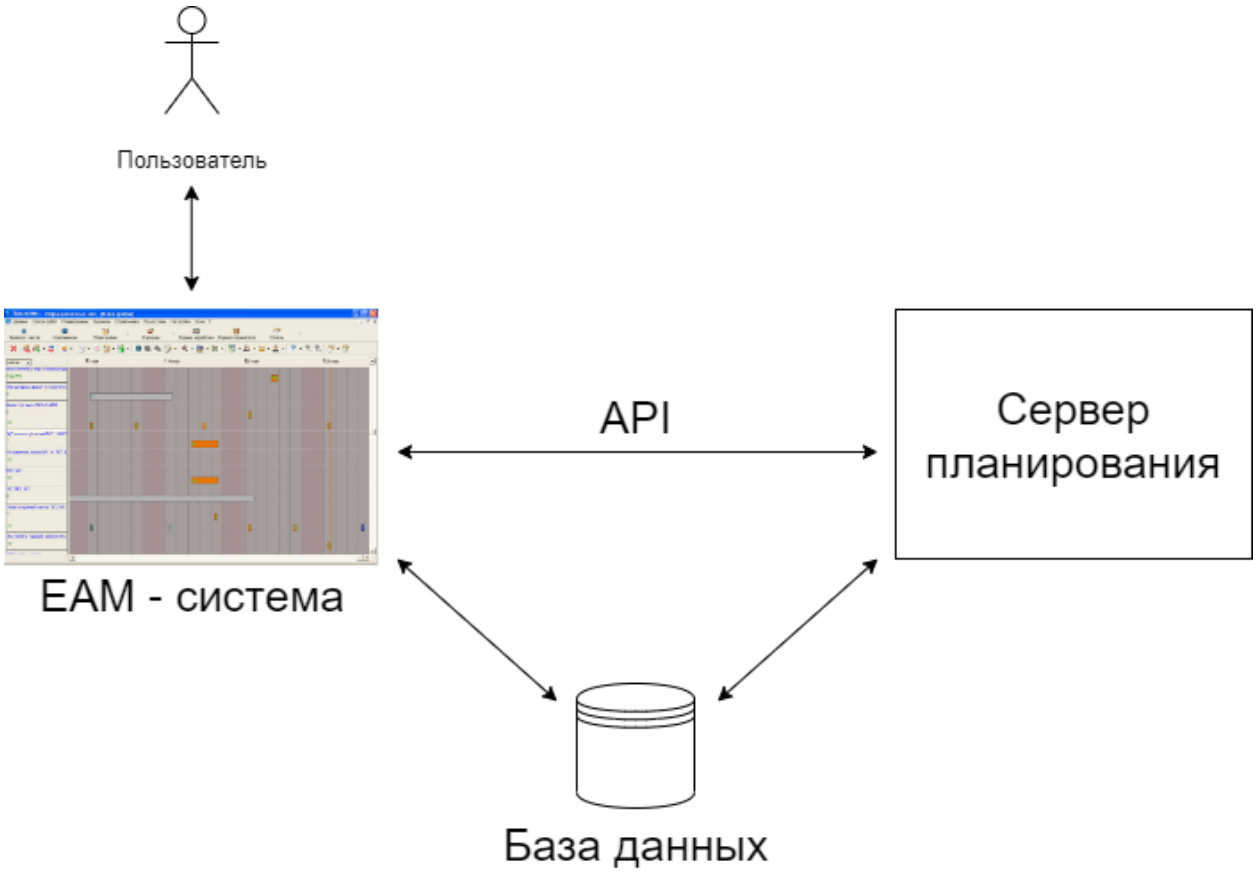

Рисунок 1 – Архитектура системы.

Архитектуру самого сервера планирования (рис. 2) можно представить как API, который посредством контроллера вызывает сервис перепланирования, использующий библиотеку OR-Tools.

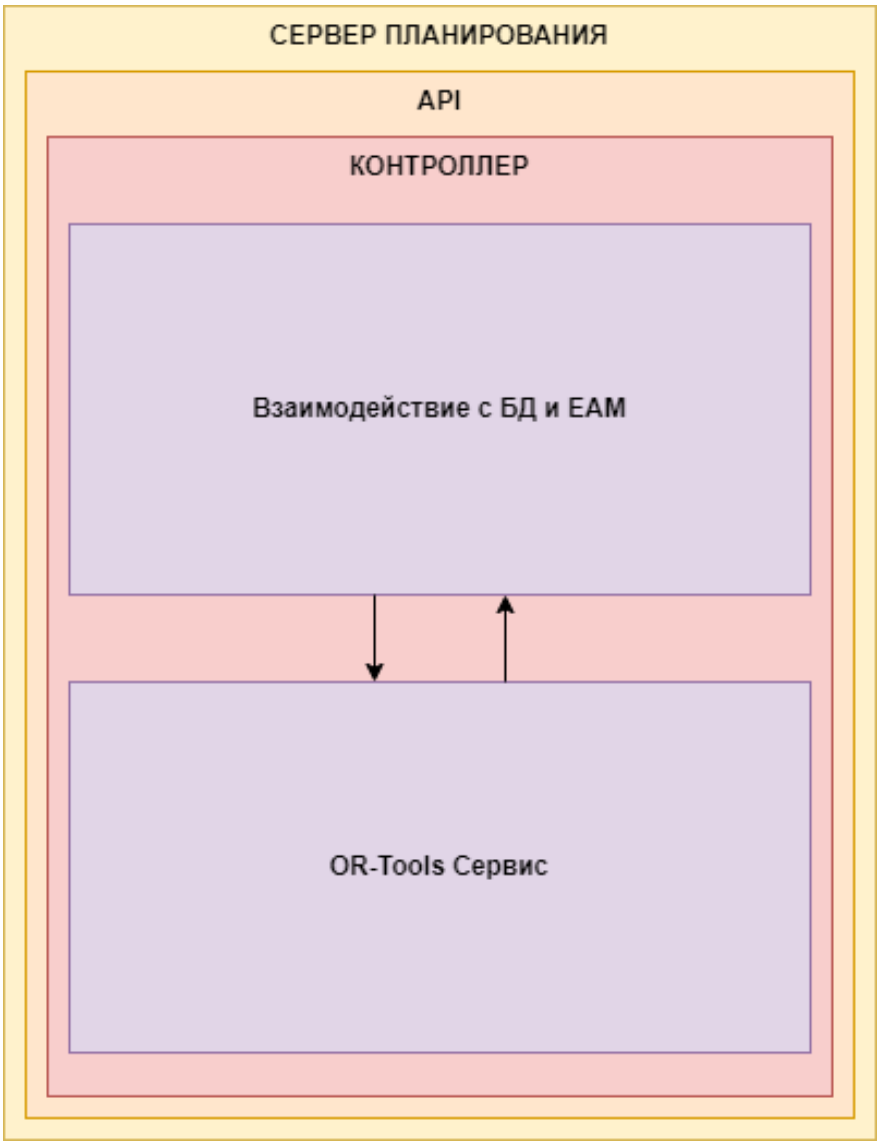

Рисунок 2 – Архитектура сервера планирования.

Рассмотрим представленную на рисунке 2 архитектуру сервера планирования более подробно.

Фронтенд (EAM-система) отправляет команду через API, в котором в привязанном к маршруту контроллере выполняются определенные действия - инициализация перепланирования выбранных через фронтенд работ или их же, но уже перепланированных, отправка обратно во фронтенд (EAM-систему) на отображение и подтверждение.

Фактически самой главной частью представленного сервера планирования является сервис OR-Tools. Именно данный сервис отвечает за основную часть - непосредственно процесс планирования предварительно задан-

ных работ. В данном сервисе происходит инициализация модели, задание ограничений для этой модели, а также задается целевая функция, то есть функция, относительно которой будут искаться все удовлетворяющие ограничениям решения для данной модели.

2.2 Разработка прототипов алгоритмов, структур данных.

2.2.1 Типы данных Google OR-Tools

Все объекты, для которых проводится процесс оптимизации, должны быть представлены в фреймворке Google OR-Tools одним из следующих типов данных:

- IntVar;
- BoolVar;
- IntervalVar;
- OptionalIntervalVar.

Тип данных IntVar это объект, который может принимать любое целочисленное значение в определенных диапазонах, такой диапазон называется доменом. В коде объявляется как

```
var integer_variable = model.NewIntVar(0, 100,
 ↪  "Integer Varible");
```

где 0 и 100 это домен переменной, а ”Integer Variable” - произвольное имя переменной, задаваемое пользователем.

Тип данных BoolVar представляет собой переменную для хранения логических значений. Внутри фреймворка BoolVar аналогичен типу IntVar, но с доменом $[0, 1]$. Выделен в отдельный тип данных для удобства программирования.

Пример объявления:

```
var boolean_variable1 = model.NewBoolVar("Boolean
 ↪  Variable");
```

В качестве параметров при объявлении передается только произвольное имя переменной, так как домен по умолчанию лежит в пределах $[0, 1]$

Тип IntervalVar представляет собой интервальную переменную. Интервальная переменная - это и ограничение, и переменная. Она определяется тремя целочисленными переменными: start, size и end.

Она является ограничением, потому что внутри она обеспечивает выполнение того, что start + size == end.

Это также переменная, поскольку она может задаваться в определенных методах ограничения планирования, таких как: NoOverlap, NoOverlap2D, Cumulative.

Объявляется так:

```
var interval_variable = model.NewIntervalVar(start,
 ↪  size, end, "Interval Variable");
```

в которой start - предполагаемый момент начала интервала, size - размер (длина) интервала - обычно фиксированное целое число, end - предполагаемый момент окончания интервала.

В качестве значений start и end могут приниматься либо целые числа - в таком случае, например, суть оптимизации такой переменной заключается в нахождении её места на горизонте планирования, либо переменные типа IntVar - тогда оптимизация данной переменной будет зависеть и от также неизвестных переменных определенного домена.

Последним из доступных типов является OptionalIntervalVar - опциональный интервал. Этот отличается от IntervalVar литералом is_present, который указывает, активна (то есть фактически присутствует в модели и участвует в процессе оптимизации) данная переменная или нет.

Пример:

```
var optional_interval_variable =
 ↪  model.NewOptionalIntervalVar(start, size, end,
 ↪  is_present "Optional Interval Variable");
```

в которой is_present представляет собой логическая переменную типа BoolVar.

Представленные выше типы данных определяются как часть модели. Модель это некоторое условное пространство имён, которое содержит в себе переменные-объекты модели, методы для работы с объектами, ограничения для них, а также методы для задания целевой функции модели.

Решение модели эквивалентно нахождению для каждой переменной единственного значения из множества начальных значений (называемого начальной областью), такого, что модель выполнима, или оптимальна, если вы предоставили целевую функцию.

Основными ограничениями для модели можно выделить:

– Add;
– AddNoOverlap;
– AddAllDifferent;
– AddImplication.

Метод Add добавляет ограничение в виде линейного выражения вида $x + y = z$, где x, y, z - целые числа или переменные типа IntVar.

Ограничение AddNoOverlap гарантирует, что все выбранные для него интервалы не пересекаются во времени.

AddAllDifferent заставляет все переменные иметь разные значения.

Метод AddImplication обеспечивает ограничение вида ЕСЛИ, ТО - $a => b$.

Класс LinearExpr позволяет производить над типами данных OR-Tools различные математические операции, такие как сумма, скалярное произведение, умножение на коэффициент - Sum(), ScaleProd(), Term().

Для задания целевой функции фреймворк предлагает два метода - maximize и minimize. Данные методы позволяют максимизировать или минимизировать некоторые необходимые параметры модели.

2.2.2 Определение требований к модели

Так как при планировании работ периоды занятости ресурсов могут накладываться друг на друга независимо от качества планирования, то ожидается, что не все работы окажутся запланированными. Соответственно, цель работы планировщика заключается в максимизации количества запланированных работ и выражается формулой:

$$\max \sum_{0}^{N} (n_0, ..., n_N) \tag{2.1}$$

где N — количество работ;

n — переменная домена $[0, 1]$, означающая присутствие работы.

Так как работы одного типа и работы, привязанные к одному человеку, никак не должны пересекаться, то должно так же выполняться следующее ограничение:

$$\forall p \in I \ \ \forall q \in I : p \neq q \implies V_p \cap V_q \neq \emptyset \tag{2.2}$$

где V_i — семейство интервалов;

p, q — интервал множества V.

Из дополнительных ограничений следует отметить:

Для каждого рабочего количество запланированных работ должно быть меньше или равно изначальному количеству.

Так как в типах работ присутствуют плановые работы, то для работ таких типов должна соблюдаться определенная периодичность, например, планирование раз в день, неделю, месяц и так далее.

3 Реализация предлагаемого решения и его оценка

Реализация системы ресурсного планирования состоит в описании модели методами целочисленного программирования, интеграции этой модели в сервис планирования и написания подобия EAM-фронтенда, который будет взаимодействовать с сервисом, то есть отображать исходные данные, запускать процесс перепланирования, отображать результаты работы сервиса, а также различную статистику.

3.1 Реализация прототипа технического решения

3.1.1 Описание инструментов и форматов данных

В качестве среды программирования была выбрана Visual Studio 2022 под операционной системой Windows 11.

Библиотека Google Or-Tools поставляется из официального NuGet репозитория.

Как сервис ресурсного планирования, так и частичный функционал подобия EAM-системы будут реализованы с помощью Blazor Server.

Данные о ресурсах и работах будут храниться в реляционной базе данных PostgreSQL, взаимодействие с базой данных извне будет происходить через SQL клиент DBeaver, а внутри проекта через ORM Entity Framework Core.

Структурная схема использованных инструментов представлена на рисунке 3.

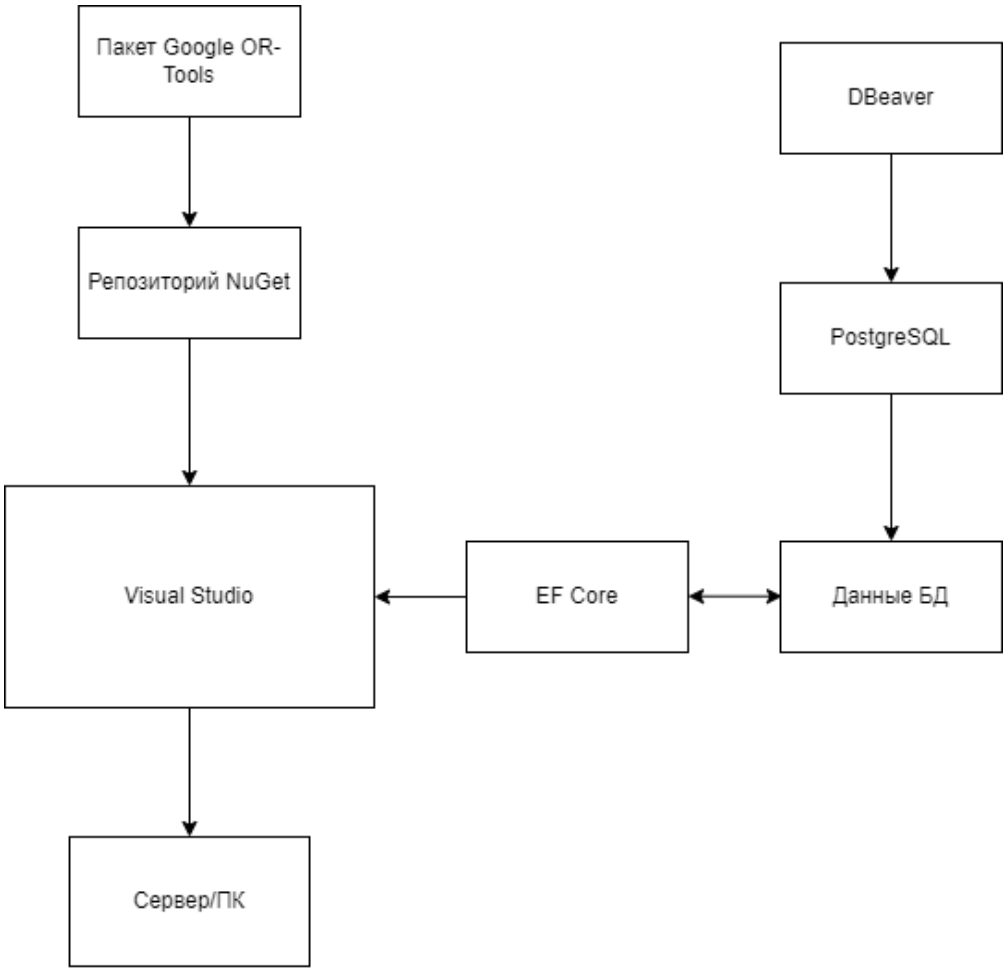

Рисунок 3 – Структура взаимосвязей инструментов.

3.1.2 Описание модели базы данных

Схема реляционной базы данных, достаточной для описания модели, представленной в главе 1 представлена на рисунке 4.

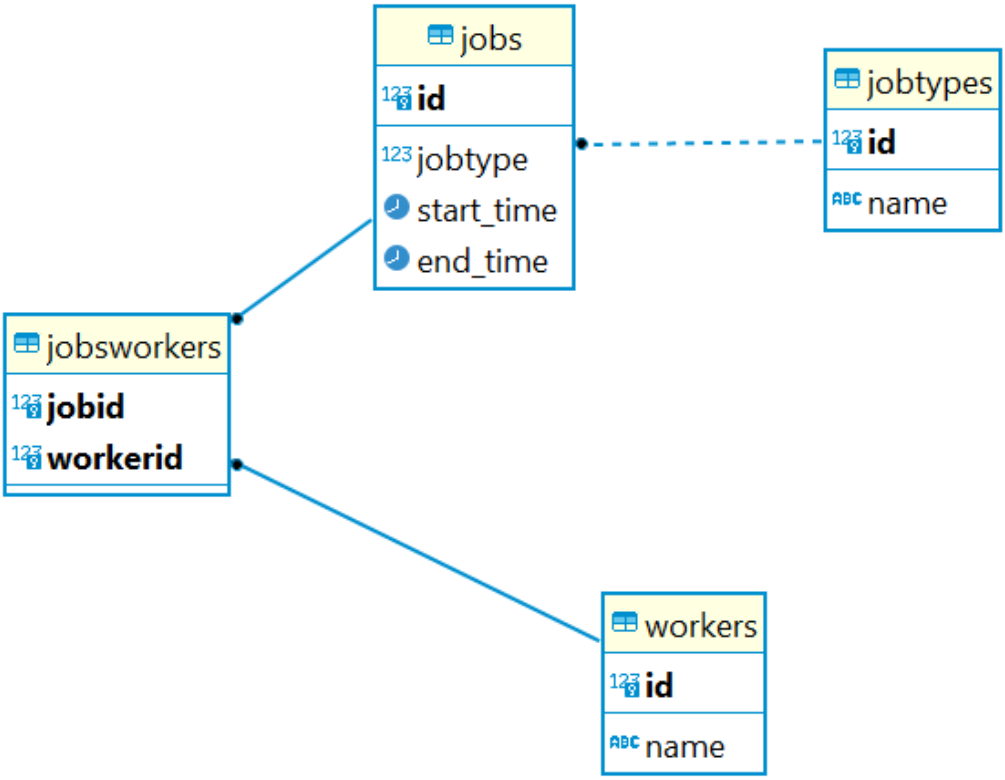

Рисунок 4 – Схема базы данных.

В схеме представлены следующие сущности:

– jobs
– jobtypes
– workers
– jobsworkers

Сущность jobs представляет все работы, запланированные на текущий момент времени.

Сущность jobtypes представляет все доступные типы работ. Связана с сущностью jobs связью один ко многим.

В таблице workers находятся все рабочие, которые могут быть привязаны к работам. Связана с таблицей jobs связью многие ко многим, поэтому необходимо использование промежуточной таблицы jobsworkers.

3.1.3 Прототипирование EAM-системы

Для упрощения взаимодействия с сервисом планирования, а так же для упрощения написания вспомогательных инструментов, в качестве платформы для EAM-системы был выбран веб-фреймворк Blazor Server.

При использовании модели хостинга Blazor Server приложение выполняется на сервере из приложения ASP.NET Core. Обновления пользовательского интерфейса, обработка событий и вызовы JavaScript обрабатываются через соединение SignalR.

Данный фреймворк позволяет создавать реактивные веб-приложения без использования языка JavaScript и его фреймворков, что значительно упрощает и ускоряет разработку веб-приложений для .Net программистов, не владеющих стеком технологий, таких как JavaScript, Angular, React и подобных.

Желаемый вид EAM-системы представлен на рисунке 5:

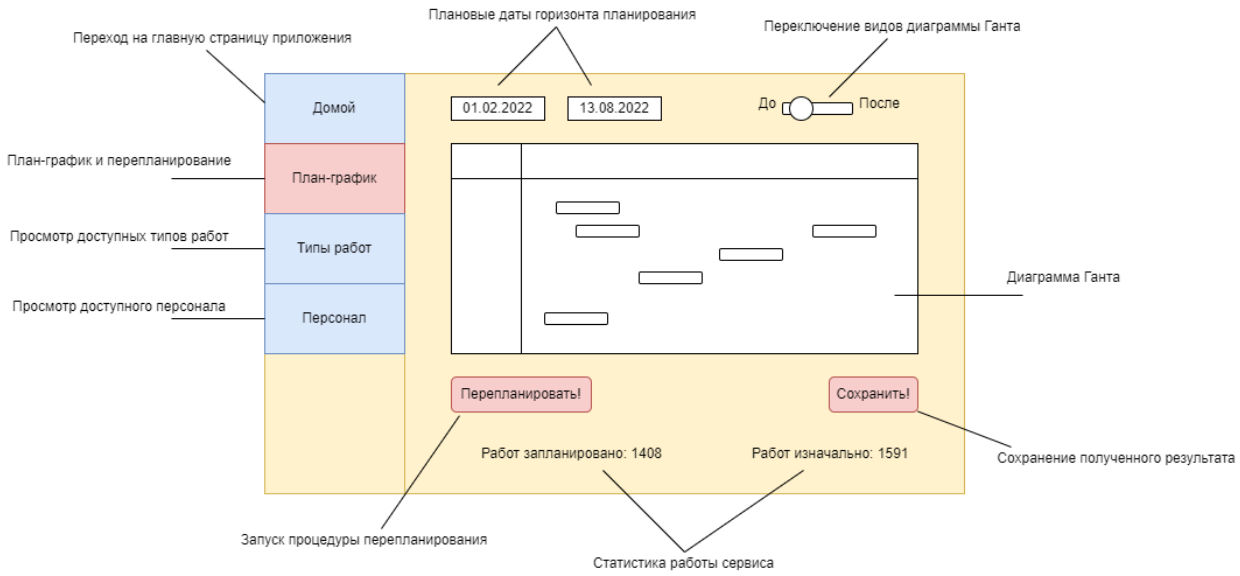

Рисунок 5 – Желаемый интерфейс прототипа EAM-системы

На рисунке показано основное окно системы - окно планирования работ, в котором находятся элементы управления для выбора плановых дат, перепланирования и записи результата в базу данных. Основным информационным элементом на данной странице является диаграмма Ганта.

Страница ”Домой” служит для отображения основной информации о прототипе системы, страницы ”Типы работ” и ”Персонал” служат только для отображения исходных данных по сущностям ”Типы работ” и ”Персонал”.

Диаграмма Ганта - столбчатая диаграмма, которая используется для иллюстрации плана или графика работ, например, во времени. Данный тип диаграмм повсеместно используется в системах подобного класса и очень удобен для отображения информации данного типа.

Так как диаграмма Ганта достаточно сложный к реализации элемент, было принято решение использовать его готовую реализацию. Готовая реализация была взята из пакета Syncfusion [20], который бесплатно предоставляет ряд веб-компонент для некоммерческого использования и для студентов. Вид диаграммы из пакета Syncfusion Blazor представлен на рисунке 6.

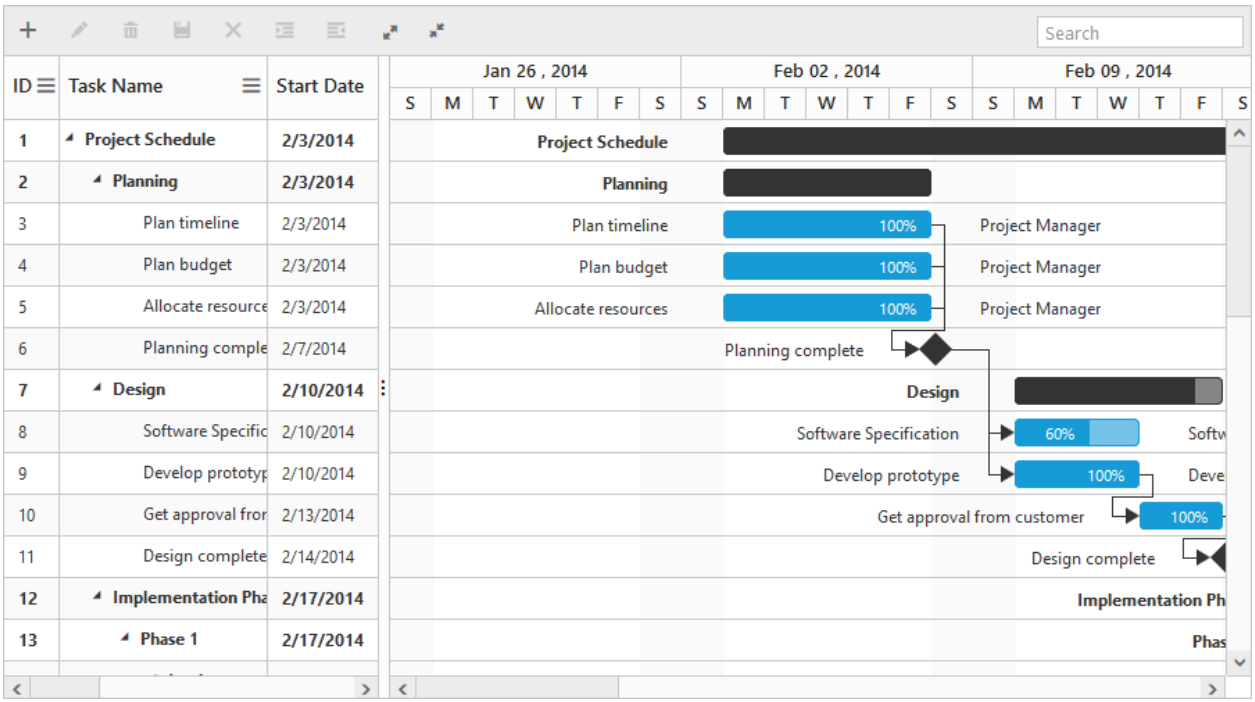

Рисунок 6 – Компонент диаграммы Ганта из пакета Syncfusion

Пакет готовых форм Syncfusion предоставляет возможность отображения работ одного типа в одной строке только в версии для фреймворка Angular, поэтому в данном прототипе каждая работа представлена отдельной строчкой на диаграмме.

Итоговый вид основного окна представлен на рисунке 7.

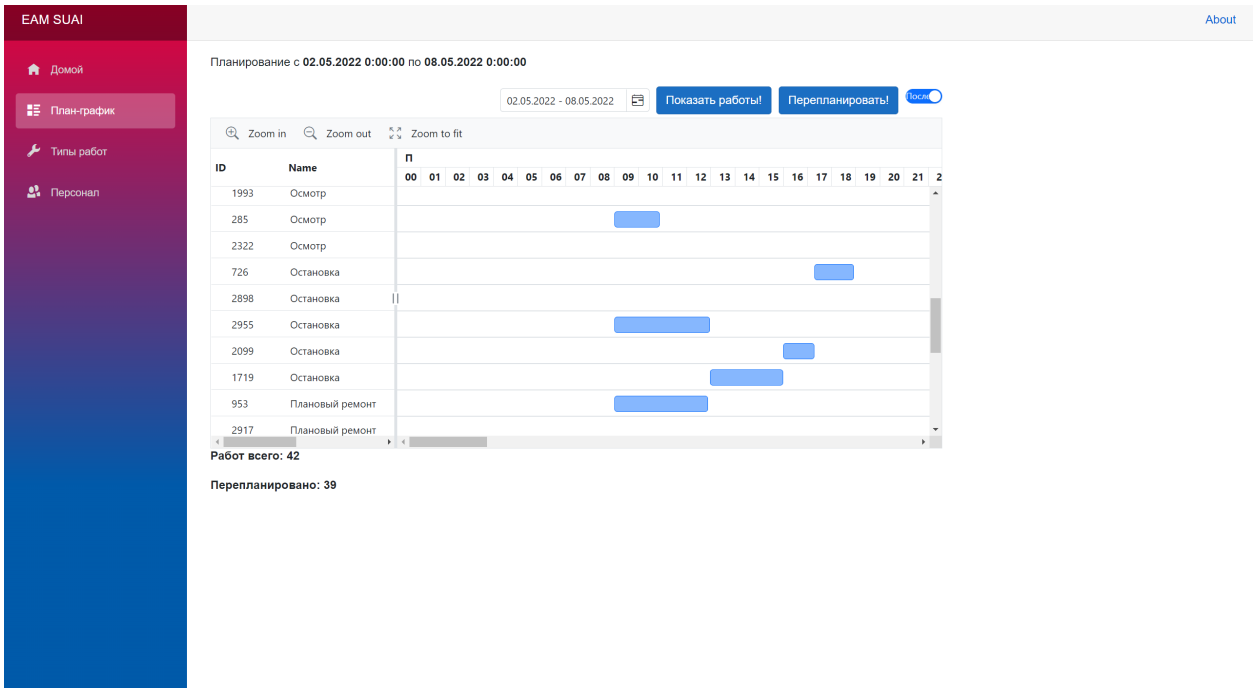

Рисунок 7 – Страница планирования работ прототипа EAM-системы

От кнопки сохранения перепланированных работ было решено отказаться, так как данный функционал не является необходимым для выполнения данной работы.

Остальные страницы прототипа представлены на рисунках 8, 9 и 10.

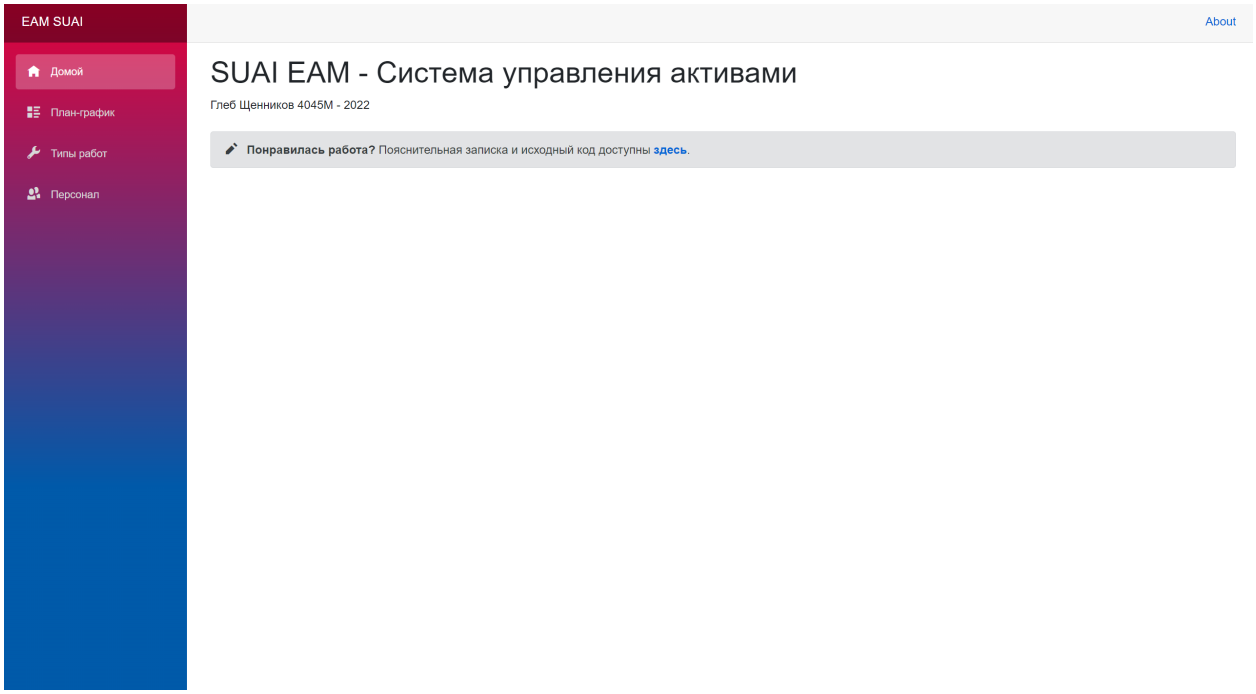

Рисунок 8 – Домашняя страница прототипа

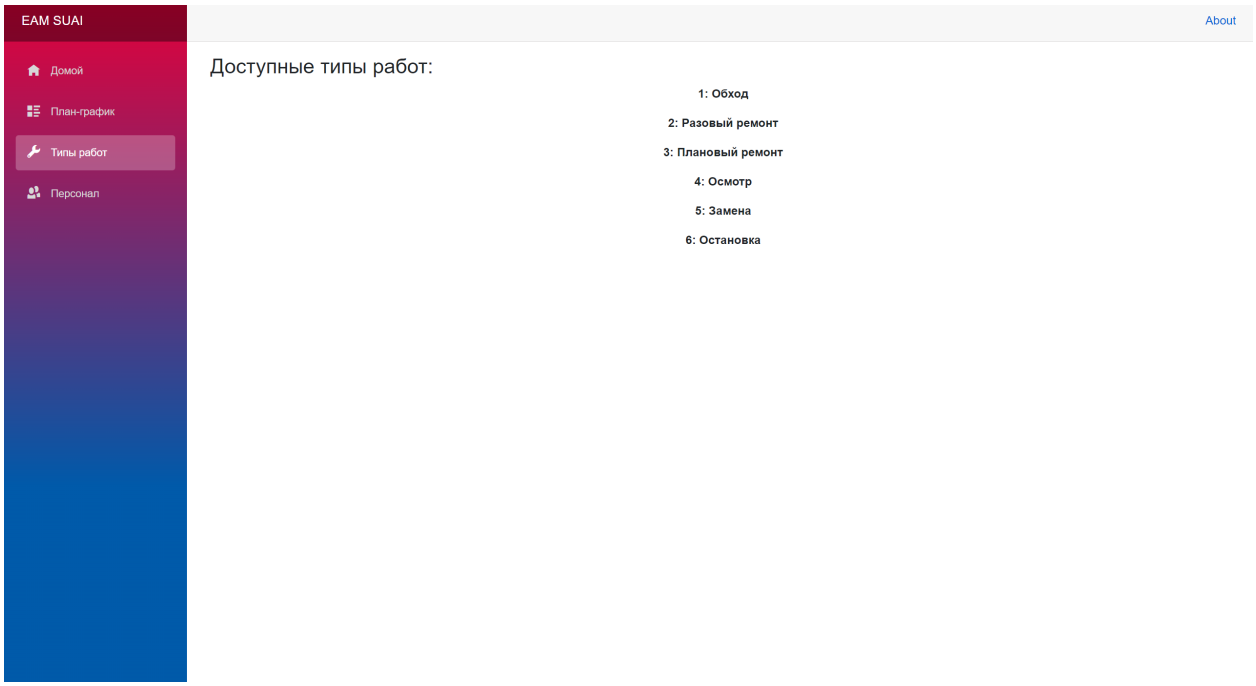

Рисунок 9 – Страница ”Типы работ”

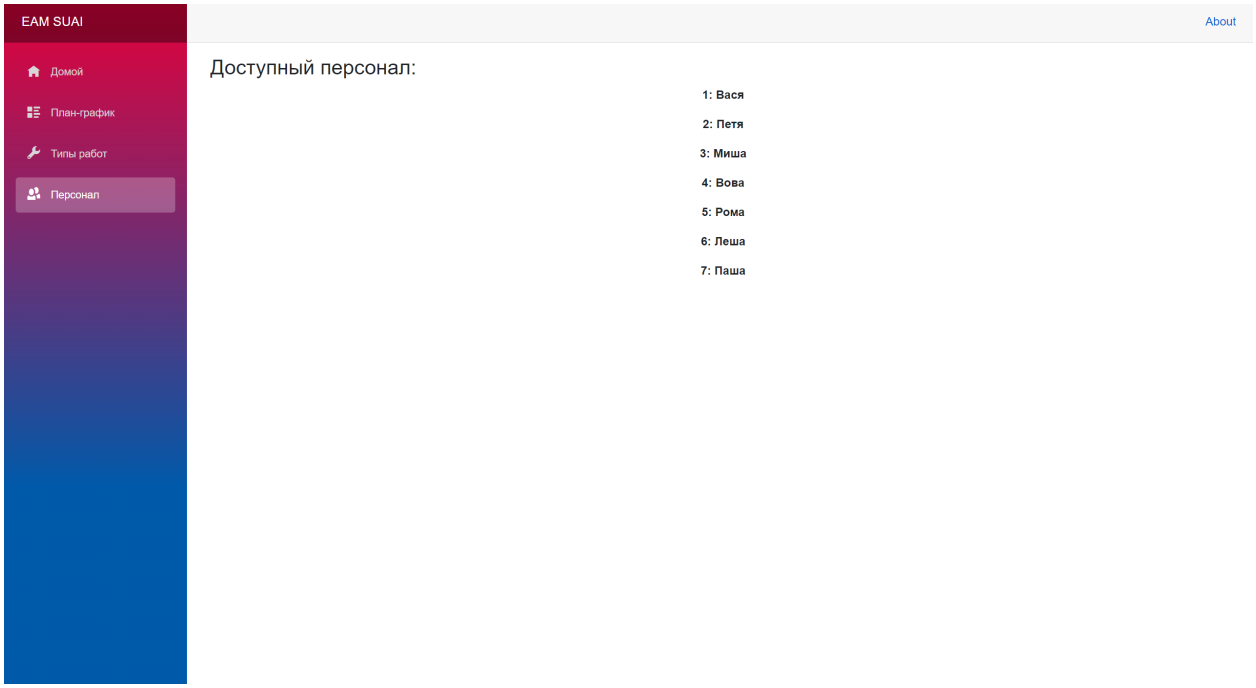

Рисунок 10 – Страница ”Персонал”

Главным преимуществом использования фреймворка Blazor в подобных проектах является отсутствие необходимости в знании каких-либо дополнительных языков и их средств, например, JavaScript и React - вся реализация веб-приложения состоит целиком из HTML/CSS и кода на языке C#.

В качестве примера можем рассмотреть страницу планирования работ. Данная страница состоит из двух частей: Razor страницы/компонента с расширением .razor или .cshtml и вспомогательного кода, наподобие модели-представления из паттерна MVVM. Вспомогательный код может находится либо прямо внутри представления, либо отдельным файлом как составной класс данного представления. Такая структура представлена на рисунке 11.

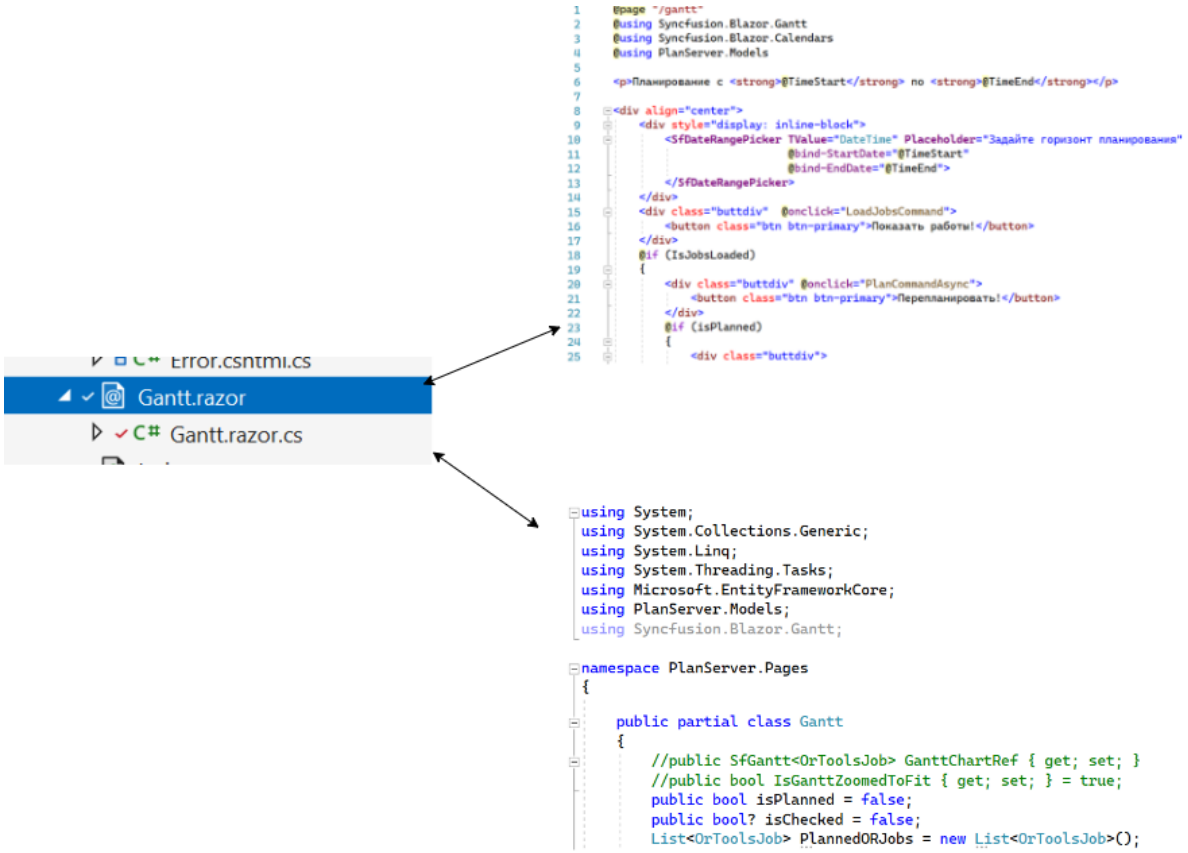

Рисунок 11 – Страница ”Персонал”

Как видно из рисунка выше, в представлении Gantt.razor задан HTML-код со вставками на C#. Данные вставки берутся из методов и свойств составного класса Gantt.razor.cs и позволяют реактивно менять значения на странице.

3.1.4 Моделирование системы средствами программирования в ограничениях

Все действия по планированию и/или перепланированию работ происходят на горизонте планирования, который выбирается пользователем и в рамках данной работы определятся целочисленным типом данных. Горизонт планирования рассчитывается в минутах и находится в диапазоне от 0 до 2,147,483,647 (ограничивается целочисленным типом данных языка C#). Такой диапазон покрывает 4085 лет и является более, чем достаточным для всех реальных задач планирования в EAM-системах. Был выбран горизонт планирования в минутах, так как он обеспечивает достаточную точность относительно часов, а в численном виде значительно меньше секунд, что сократит необходимую память для планирования.

Для описания работ были выбраны следующие типы данных, доступные из библиотеки OR-Tools:

– BoolVar
– IntVar
– OptionalIntervalVar

Тип данных IntVar в данной модели определяет переменные моментов начала и конца работ. Лежит в диапазоне от 0 до значения горизонта планирования.

Типом BoolVar определяется актуальность работы, то есть ее участие в процессе планирования. Данная переменная используется в типе OptionalIntervalVar. Переменная задается в диапазоне от 0 до 1 по умолчанию, соответственно если значение переменной равно единице, то работа учитывается при планировании, 0 - не учитывается.

OptionalIntervalVar уже определяет саму работу на горизонте планирования. В отличие от просто IntervalVar данный тип - опциональный и зависит от переменных типа BoolVar, используемых литералом is_present.

Для составления модели до момента задания ограничений необходимо в первую очередь инициализировать работы, то есть инициализировать все переменные, представляющие работы.

В первоначальной версии модели инициализация была проделана так:

```
for h in range(horizon)
    ...
    for w in workers
        ...
        for jt in job_types
            ...
            for job in jobs
```

В данном варианте инициализация происходит относительно четырех сущностей модели: каждой минуты горизонта планирования, каждого рабочего, каждого типа работы и непосредственно каждой работы.

Отсюда, например, при 10 рабочих, 10 типах работ, 10 запланированных работах и горизонте планирования в 10080 минут (семь дней):

$10 * 10 * 10 * 10080 = 10080000$

Получаем некий объект, в котором уже для простейшего примера ресурсного планирования получим непомерно огромные затраты по памяти, причем полученную цифру в 10080000 необходимо еще умножить на 4, так как имеем при инициализации для каждой работы две переменных типа IntVar, одну BoolVar и одну OptionalIntervalVar, в итоге - свыше сорока миллионов объектов для простого тестового примера.

Необходимость в наличии инициализации относительно горизонта планирования была связана с недопониманием концепции использования опциональных интервалов. Изначально казалось необходимым наличие возможности начала работы в каждую из минут горизонта планирования, что соответственно добавляло лишнее измерение в инициализацию.

Тем не менее, такой подход оказался рабочим и выдавал удовлетворяющий результат, то есть соответствовал ограничениям и целевым функциям модели, откуда можем сделать вывод о возможности наличия более чем одного подхода в моделировании при решении проблем программирования в ограничениях.

Результаты исследований прототипа модели первой версии будут представлены ниже в соответствующей главе.

Во второй итерации модели, уже при переносе с языка Python на C#, были найдены некоторые недочеты первой версии модели, а также пересмотрена процедура инициализации, в результате которой был убран цикл по горизонту планирования. Это обуславливается тем, что в интервальных переменных и так задается их момент возможного начала и момент возможного конца, соответственно нет необходимости в привязке начала интервалов к конкретному моменту горизонта планирования.

Итоговый вид инициализации на языке C#:

```
...
foreach (var w in workers)
{
    ...
    foreach (var jt in job_types)
    {
    ...
        foreach (var job in jobs)
        {
            if (jt == job.Type && w == job.WorkerID)
            {
                var performed_at =
↪   model.NewBoolVar($"performed by {w}wID {jt}type
↪   {job.ID}jID");
                var start = model.NewIntVar(0, horizon,
↪   $"start of {w} {jt} {job.ID}");
                var end = model.NewIntVar(0, horizon,
↪   $"end of {w} {jt} {job.ID}");
```

```
                var optint =
 ↪  model.NewOptionalIntervalVar(start,
 ↪  job.TimeSpanMinutes, end, performed_at, $"interval
 ↪  of {w} {jt} {job.ID}");
                ...
            }
        }
```

Следующим шагом после инициализации входных данных для модели является задание ограничений.

Первая группа ограничений - работы одного типа, а так же работы одного рабочего не должны пересекаться.

Для этого языком подзапросов LINQ из переменных, представляющих входные данные необходимо достать необходимые работы. Для рабочих:

```
//работы по рабочему
List<List<IntervalVar>> jobIntervals_by_worker =
 ↪  new();
foreach (var w in intervals_by_worker)
{
    var flatlist = w.SelectMany(i => i).ToList();
    jobIntervals_by_worker.Add(flatlist);
}

//работы одного рабочего не пересекаются
foreach (var ji in jobIntervals_by_worker)
{
    ji.AddRange(sleepIntervals);
    model.AddNoOverlap(ji);
}
```

В данном примере следует обратить внимание на метод AddNoOverlap, который и устанавливает ограничение на непересечение элементов выборки ji, то есть множества работ одного рабочего.

Задание ограничений этой группы для работ одного типа происходит аналогично.

Дополнительно следует отметить, что для множества рабочих было задано дополнительное ограничение в виде работы только по будням с девяти утра до девяти вечера.

```
    //определим нерабочие (ночные интервалы) -
↪   допустим с 9 утра до 9 вечера
    //даты планирования выбираются пользователем с
↪   00:00 по 00:00
    //0 + 540 => ночь; +720 => день; + 180 => остатки
↪   ночи
    List<IntervalVar> sleepIntervals = new();
    int dayCounter = 0;

    foreach (DateTime day in EachDay(reftime, lastday))
    {
        int st = dayCounter * 1440;
        if ((day.DayOfWeek == DayOfWeek.Saturday) ||
↪   (day.DayOfWeek == DayOfWeek.Sunday))
        {
            //по выходным не работаем
            int fin1 = st + 1440;
            var siSS = model.NewIntervalVar(st, 1440,
↪   fin1, $"sat/sun {dayCounter}");
            sleepIntervals.Add(siSS);
        }
        else
```

```
        {
            //по будням - с 9 до 9
            int fin = st + 540;
            var si = model.NewIntervalVar(st, 540, fin,
↪   $"sleep 1 of day {dayCounter}");
            sleepIntervals.Add(si);
            st = fin + 720; //пропускаем рабочие 12
↪   часов
            fin = st + 180;
            //второй ночной интервал
            var si2 = model.NewIntervalVar(st, 180,
↪   fin, $"sleep 2 of day {dayCounter}");
            sleepIntervals.Add(si2);
        }
        dayCounter++;
    }
```

Следующим шагом необходимо ограничить возможное количество работ. В данном варианте мы считаем, что перепланированное количество работ одного типа и количество работ у одного рабочего должно быть меньше или равно исходному их количеству. Тогда для работ одного типа:

```
    ///разделяем performed_at по типам работы
    Console.WriteLine("разделяем performed_at по типам
↪   работы");
    List<List<IntVar>> performedAt_by_type = new();
    foreach (var t in job_types)
    {
        List<IntVar> perflist = new();
        performedAt_by_type.Add(perflist);
    }
    foreach (var b_worker in bools_by_worker)
```

```
    {
        for (int b_type = 0; b_type < b_worker.Count;
↪   b_type++)
        {
            var flatperf = b_worker[b_type];

↪   performedAt_by_type[b_type].AddRange(flatperf);
        }
    }
    for (int i = 0; i < performedAt_by_type.Count; i++)
    {

↪   model.Add(LinearExpr.Sum(performedAt_by_type[i])
↪   <= job_type_num[i]);
    }
```

Необходимое ограничение задается в строке `model.Add(LinearExpr.Sum(performedAt_by_type[i]) <= job_type_num[i])`, где метод Add() задает для модели некое линейное ограничение. Метод LinearExpr.Sum(), переданный в Add() позволяет производить над множествами объектов типа OR-Tools операцию сложения значений элементов множества. Соответственно данной строкой мы задаем для модели, что количество работ перепланированных меньше или равно количеству работ исходных.

Ограничение по количеству работ для конкретных рабочих задается аналогично.

Так как у работ может быть задана некая периодичность выполнения, например, каждый день, раз в неделю, раз в месяц и так далее, то необходимо задать механизм эту периодичность устанавливающий.

Изначально данное ограничение представлялось очень простым. У каждой работы (опционального интервала) заданы переменные start и end

типа IntVar, обозначающие возможный момент начала и возможный момент окончания работы. Соотвественно ограничение выглядело тривиально: $start_{n+1} - end_n > X$, где X - интервал, через который плановые работы одного типа должны выполняться.

Так как переменные start и end используются в опциональном интервале, то ожидалось, что при невозможности для двух работ соответствовать ограничению $start_{n+1} - end_n > X$ - данные работы просто не будут учитываться, так как привязаны к типу OptionalIntervalVar. В действительности же заданные ограничения при невозможности их удовлетворить делали всю модель невыполнимой, в связи с тем, что ограничения к данным переменным рассматриваются сами по себе и без привязки к другим типам в модели. Именно для таких случаев используются связывающие ограничения.

Связывающие ограничения (от англ. - channeling constraint) связывают переменные внутри модели. Они используются, когда необходимо выразить сложную связь между переменными, например, ”если эта переменная удовлетворяет условию, то она заставляет другую переменную принять определенное значение”.

В данном случае необходимо указать, что если ограничение $start_{n+1} - end_n > X$ выполнимо, то литерал is_present, отвечающий за актуальность опционального интервала, станет равным единице, то есть работа с этим литералом будет учитываться при планировании.

Выражается таким образом:

```
//для третьего типа (плановая работа) - работы не чаще
↪  раза в день
    ...
    if(type == 3)
    {
        for (int i = 0; i < startAt_by_type[k].Count -
↪  1; i++)
        {
```

```
            for (int j = i+1; j <
↪   startAt_by_type[k].Count; j++)
            {
                //if (b-a>1440) => b=0
                model.Add(startAt_by_type[k][j] -
↪   endAt_by_type[k][i] >=
↪   1440).OnlyEnforceIf(performedAt_by_type[k][i]);
            }
        }
        break;
    }
    ...
```

В этом примере с помощью метода OnlyEnforceIf() устанавливается, что если выполнятся ограничение в методе Add(), то переменная типа BoolVar внутри метода OnlyEnforceIf() обратится в 1.

Последним шагом остается задать целевую функцию. В качестве целевой функции предлагается максимизировать количество перепланированных работ в целом. Для этого напишем такой код:

```
//возьмем множество всех работ
var flatbools = bools_by_worker.SelectMany(i =>
↪   i).SelectMany(i => i);
//целевая функция - максимизировать количество работ
model.Maximize(LinearExpr.Sum(flatbools));
```

В данном примере целевая функция задается через метод модели Maximize(), в который передается множество всех работ. В данном случае каждый объект элемент множества является типом данных BoolVar.

3.2 Экспериментальные исследования прототипа решения

После создания прототипа была проведена экспериментальная оценка пригодности системы для задач ресурсного планирования в соответствии с поставленной задачей.

Основные характеристики аппаратно-программной конфигурации, которая использовалась для создания прототипа: компьютер – AMD Ryzen 9 5900HX, видеокарта Nvidia GeForce RTX 3070 Laptop 8Гб, оперативная память 16Гб DDR4.

Основными критериями для исследований предлагается использовать:

– Зависимость времени планирования от количества работ/ресурсов
– Зависимость требуемой памяти от количества работ/ресурсов
– Результативность алгоритма (процент количества перепланированных работ от исходного)

3.2.1 Сравнение двух версий модели

Для начала рассмотрим время планирования (рисунок 12) и требуемый объем памяти (рисунок 13) для планирования 13 работ на горизонте в одну неделю для моделей первой и второй версии:

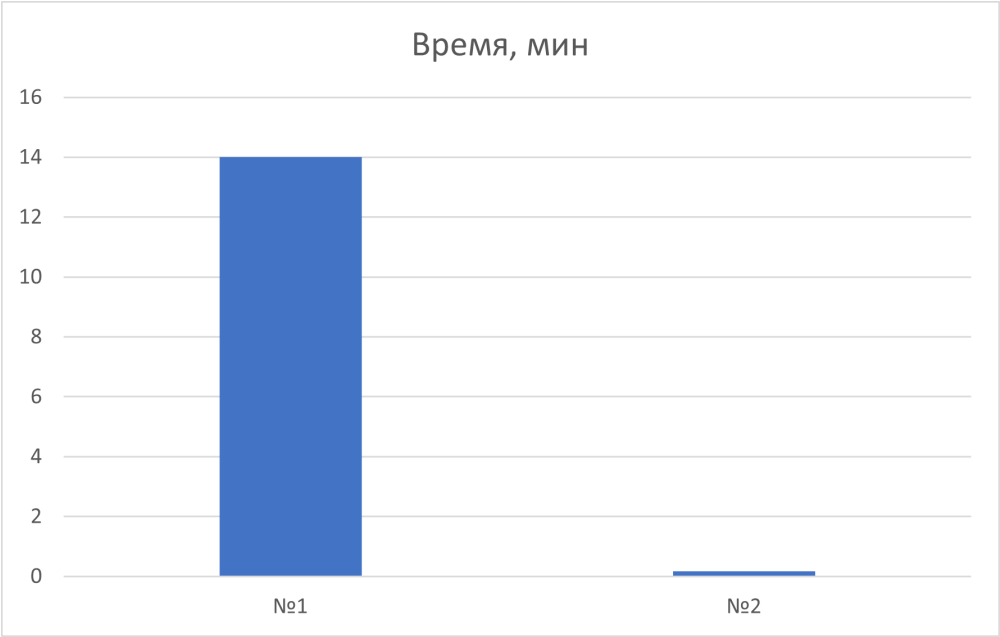

Рисунок 12 – Разница во времени планирования для моделей первой и второй версии (меньше - лучше)

Рисунок 13 – Разница в требуемой оперативной памяти для моделей первой и второй версии (меньше - лучше)

По графикам, представленным на рисунках 12 и 13, видно, что модель второй версии отрабатывает на порядок быстрее и требует на порядок меньше памяти. Время перепланирования первой модели составило 14 минут, второй модели - 5 секунд, для работы первой модели потребовалось 12 гигабайт оперативной памяти, а для работы второй - всего лишь 300 мегабайт.

Такая явная разница в результатах работы обусловлена, как было указано выше в части 3.1.4, наличием лишнего измерения при инициализации данных вследствие применения ошибочного подхода при моделировании.

Представленные выше графики показывают, что хотя результаты работы разных вариантов модели для одной и той же задачи могут быть удовлетворительными, далеко не все из них могут быть применимы в реальных условиях, так как по тем или иным причинам могут не соответствовать требованиям по памяти или быстродействию.

3.2.2 Анализ работы итоговой версии модели

Рассмотрим результаты работы второй (итоговой) версии модели.

В первую очередь проверим перепланирование в граничных условиях, для которых заранее известен результат планирования в соответствии с постановкой задачи.

Так как при моделировании были использованы нестрогие ограничения, мы точно можем сказать, что количество запланированных работ будет больше нуля при любых начальных условиях. Единственным вариантом, в котором представляется отсутствие перепланированных работ является вариант, где выбранный горизонт планирования меньше времени наименьшей работы. Такой вариант является нереалистичным и рассматриваться не будет.

Для тестирования работы в граничных условиях приведём вариант планирования работ нескольких типов на горизонте одного дня, где все работы назначены одному человеку и не могут выполниться в полном объёме не пересекаясь (рис. 14).

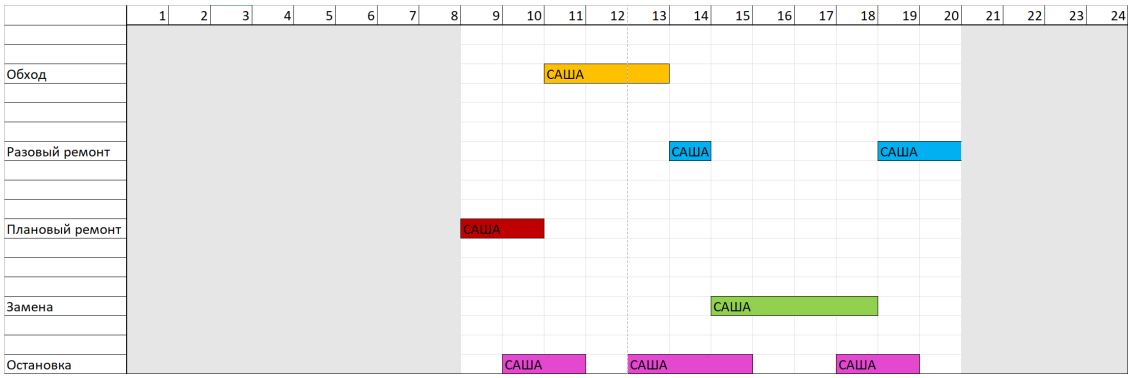

Рисунок 14 – Заведомо неисполнимый вариант планирования потребности в работах

По рисунку выше видно, что есть пять работ типов ”Обход”, ”Ремонт”, ”Плановый осмотр”, ”Замена”, которые явно следуют друг за другом и не пересекаются. Работы типа ”Остановка” находятся в границах других работ и выполнены быть не могут.

Очевидно, что исходя из данной диаграммы Ганта как минимум пять работ должны остаться после перепланирования. Есть вероятность, что при данных начальных условиях работы можно перепланировать так, что их возможное непересекающееся множество будет состоять более чем из пяти

интервалов, но уже даже на такой небольшой выборке видно, что для человека задача становится достаточно нетривиальной и потребует относительно долгое время на разрешение.

Результат перепланирвоания представлен на рисунке 15.

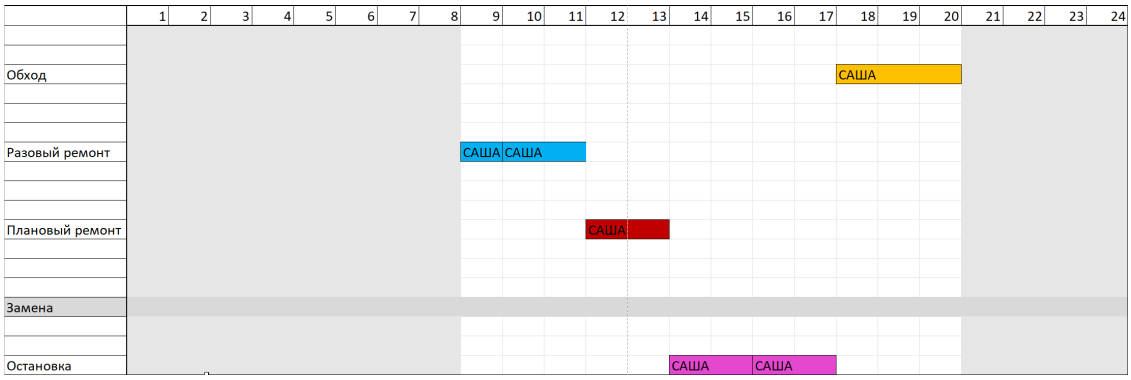

Рисунок 15 – Результат работы планировщика в граничных условиях

Как видно из рисунка 15, планировщик исключил самую длинную работу по замене, оставив вместо нее две работы по остановке, длинна которых эквивалентна работе по замене, тем самым увеличив количество запланированных работ. Непосредственно процедура перепланирования заняла меньше 1 секунды и потребовала 250 мегабайт оперативной памяти.

В интерфейсе прототипа EAM-системы результат перепланирования (рис. 16) выглядит так:

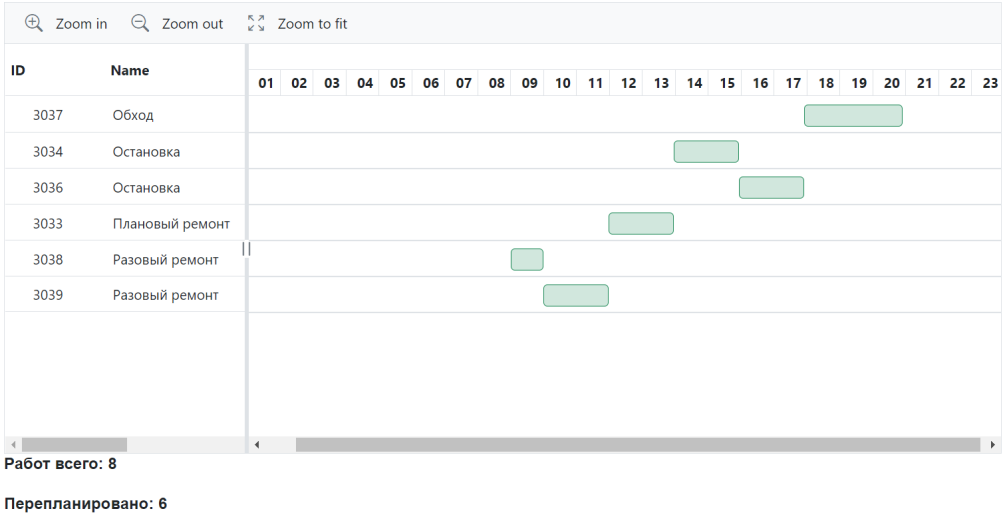

Рисунок 16 – Результат работы планировщика в граничных условиях в интерфейсе EAM-системы

В качестве второго граничного условия определим 100 случайно заданных работ на горизонте планирования в одни сутки.

По результатам тестирования в данных условиях перепланировать удалось лишь 51 из 100 работ. Визуальный осмотр результатов, сравнение диаграмм Ганта до и после, просмотр логов планировщика не выявили аномальных или подозрительных результатов, а перепланированные интервалы соответствовали поставленной задаче, поэтому можно сделать вывод, что в данном граничном эксперименте модель также отработала корректно.

При таких входных данных наиболее интересным для исследования оказался параметр времени перепланирования при разном количестве задействованных на алгоритм ядер процессора.

На рисунке 17 показана зависимость времени планирования от количества используемых процессоров.

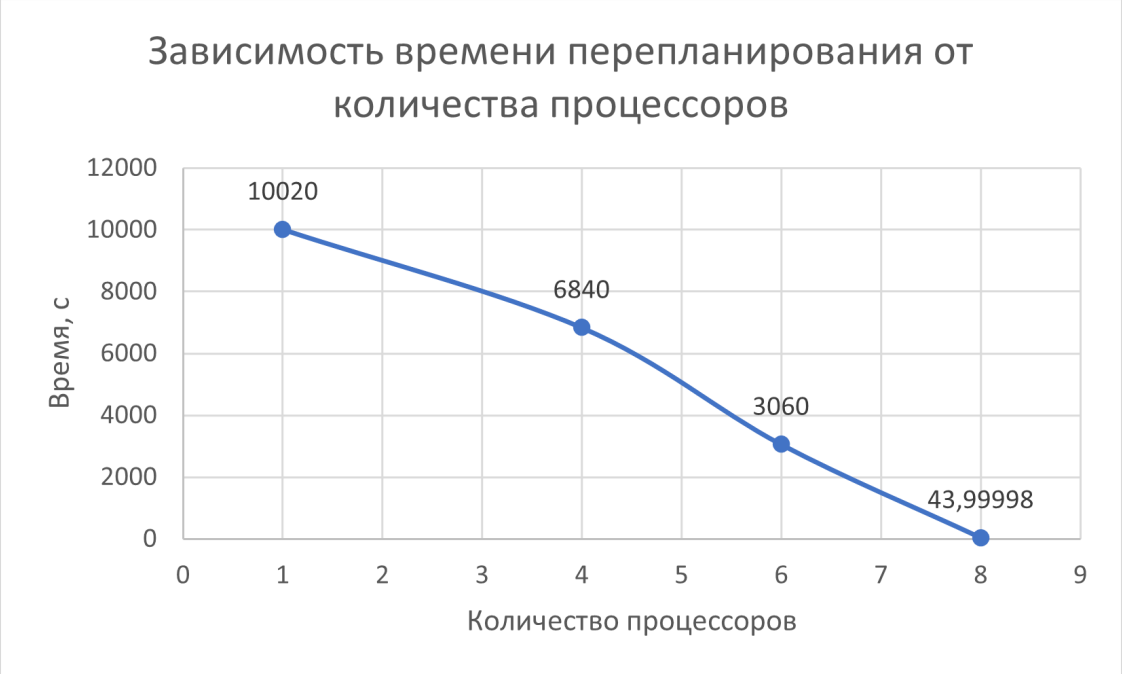

Рисунок 17 – Зависимость времени планирования от количества задействованных ядер процессора

По результатам эксперимента выше видим, что количество задействованных процессоров является одним из самых важных факторов быстродей-

ствия в данной модели. При использовании множества процессоров, каждый из них выполняет свою часть по поиску решений из множества всех решений.

Анализируя результат эксперимента в граничных условиях можно сказать, что планировщик корректно отработал в рамках поставленной задачи, показав высокую скорость работы и относительно небольшие требования к оперативной памяти. При этом, ключевую роль в быстродействии алгоритма перепланирования сыграло количество задействованных процессоров, показавшее явный, фактически линейный тренд на уменьшение времени планирования при увеличении количества процессоров.

Рассмотрим зависимость метрик перепланирования от количества работ и горизонта планирования. Количество типов работ и рабочих для всех тестов будет оставаться неизменным и равным десяти. Типы работ, продолжительность и дата выбираются случайным образом.

На рисунках 18 и 19 показано время в секундах, требуемое для перепланирования 100 и 1000 работ на горизонте от 31 до 365 дней.

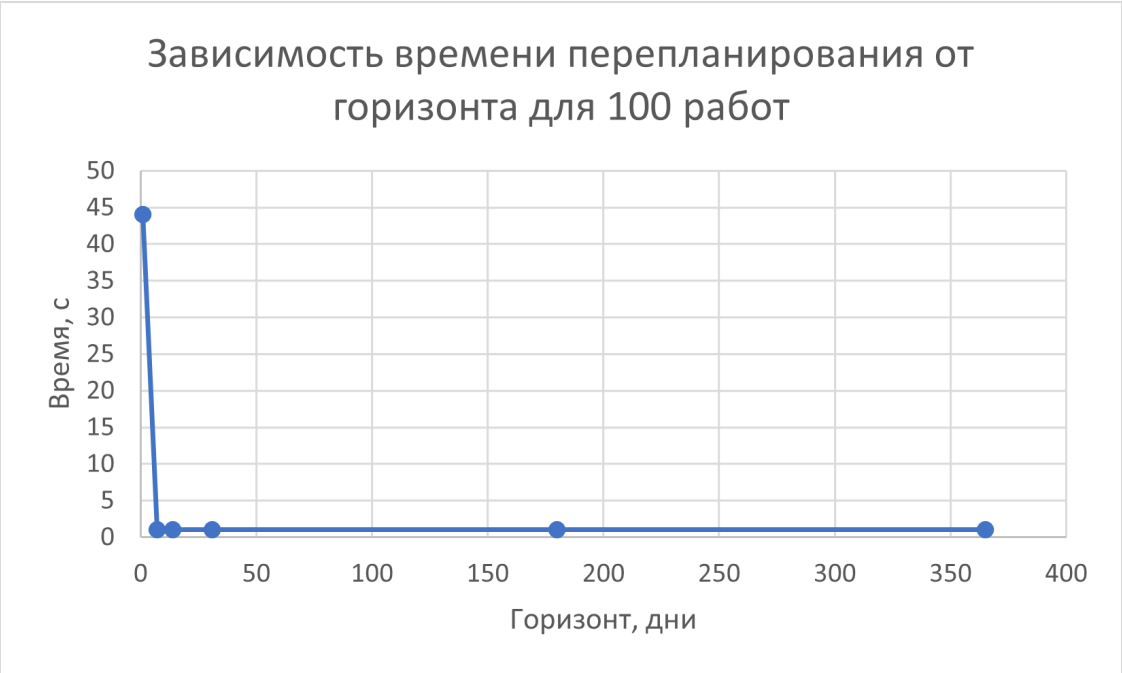

Рисунок 18 – Зависимость времени планирования от количества задействованных ядер процессора

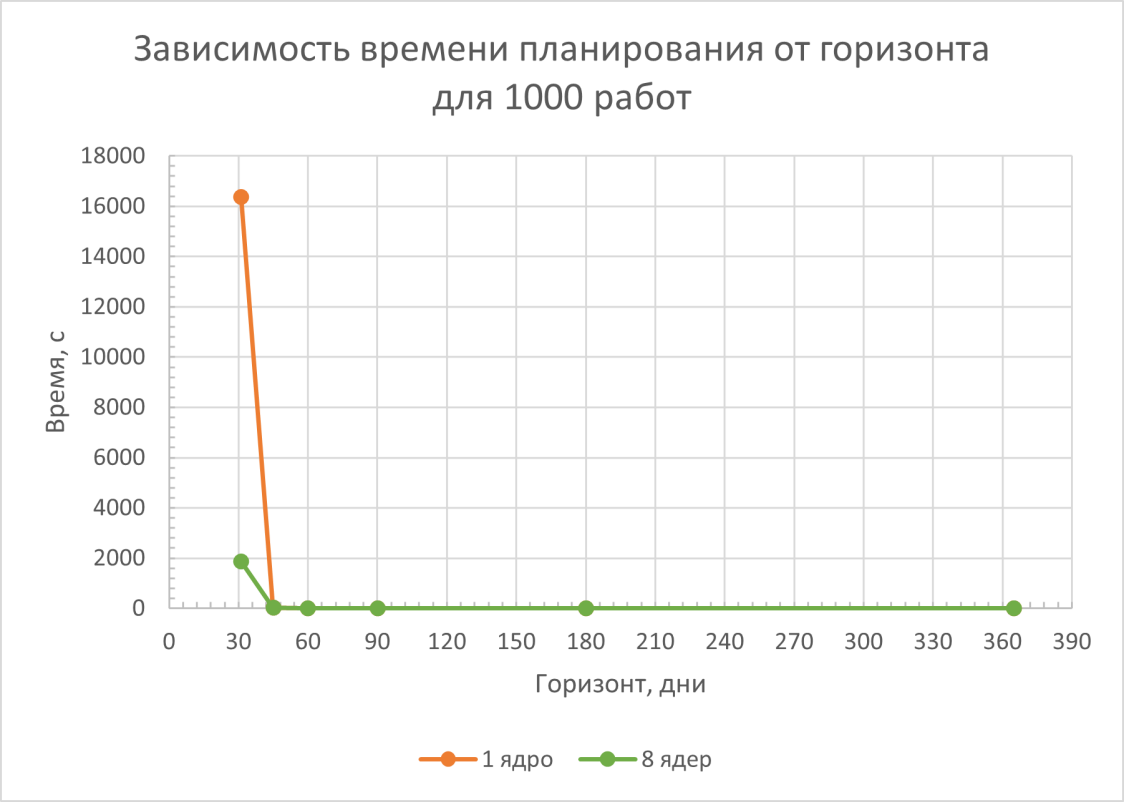

Рисунок 19 – Зависимость времени планирования от количества задействованных ядер процессора

По представленным графикам видно, что на требуемое время планирование всех работ сильно влияет выбранный горизонт планирования. Данная зависимость связана с тем, что на длительных интервалах горизонта планирования существует достаточно свободного места для всех работ, и планирование в таком случае становится достаточно тривиальным.

Соответственно, чем меньше задается выбранный интервал, тем меньше становится вариантов расположить необходимое количество работ, то есть сужается множество удовлетворяющих решений на всем возможном множестве решений.

По графику 19 сильно заметно влияние использования нескольких процессорных ядер для процесса планирования. Перепланирование с 8 ядрами оказалось примерно в 8 раз быстрее одноядерного перепланирования, что также соотносится с результатами, полученными в тестировании граничных условий (рис. 17).

Проведем замеры скорости планирования в зависимости от количества работ на горизонте в один год. Так как работы выбираются и распределяются в случайном порядке, проведем эксперимент 10 раз и построим зависимость от среднего времени перепланирования.

График зависимости времени планирования от количества работ представлен на рисунке 20:

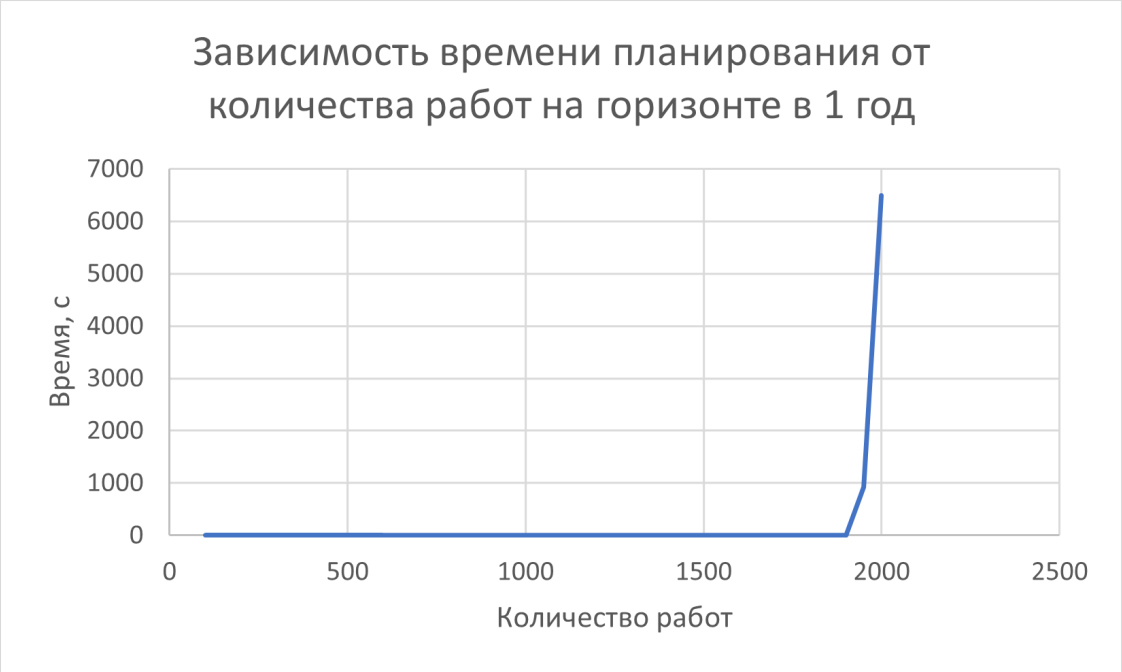

Рисунок 20 – Зависимость времени планирования от количества работ

Как видно из рисунка 20, примерное количество работ, которое в кратчайшие сроки может быть запланировано на один год находится в промежутке от 0 до 1900 работ. В процессе тестирования был выявлен промежуток, в котором перепланирование начинало занимать больше часа времени - данный промежуток составил от 1900 до 2100 работ. Среднее время перепланирования в интервале до 1900 работ составило менее 10 секунд. Увеличение горизонта планирования до двух лет позволило завершить операцию перепланирования за 3 секунды.

Было проведено исследование зависимости времени перепланирования от количества доступных рабочих. При уменьшении их количества в два раза, количество работ, которые возможно перепланировать в короткие сроки так же уменьшилось в два раза, и находится в промежутке от 800 до

1100 работ. Увеличение же числа рабочих в два раза не повлияло на время перепланирования, исходя из чего можем сделать вывод о наличии некой границы, ограничивающей быстроту перепланирования для данной модели.

Одной из интересных метрик оказалась средняя плотность запланированных работ в день (рис. 21), при которой модель справляется с перепланированием пределах двух-трех часов:

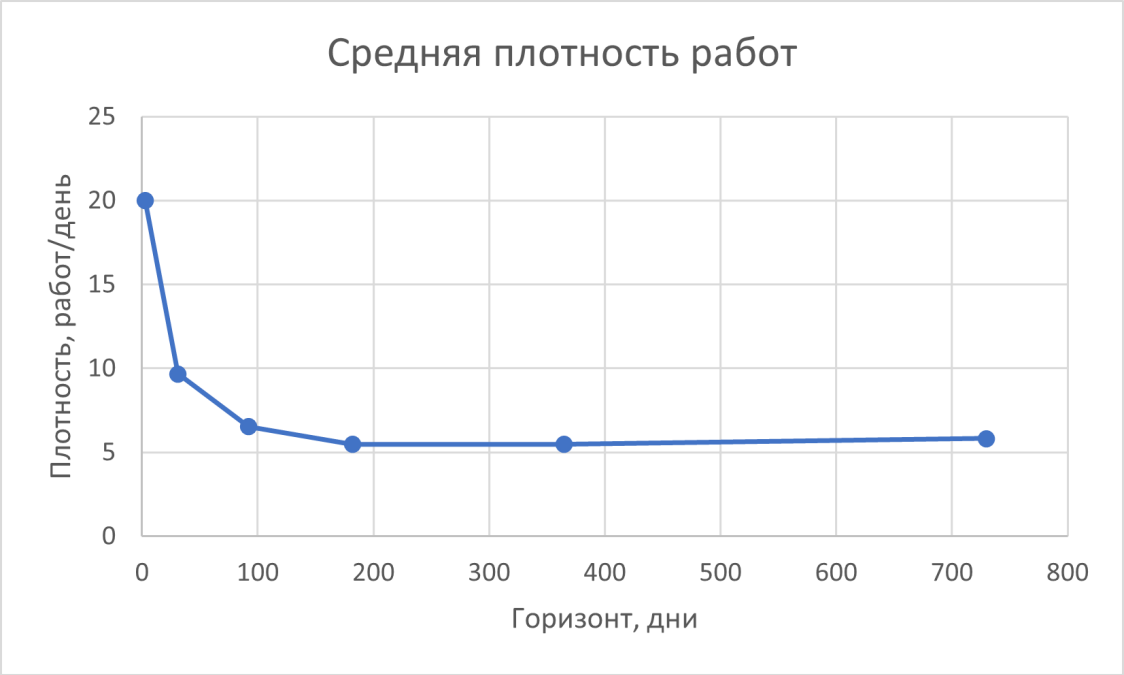

Рисунок 21 – График зависимости средней плотности планирования работ от горизонта планирования

По графику, представленному на рисунке 21 видно, что для 10 типов работ средняя максимальная плотность перепланирования составила около 5.5 работ в день при горизонте планирования от 180 дней и более. Под максимальной плотностью в данном случае понимается такая плотность, которая достигается при количестве работ, перепланируемых не более чем за три часа. Высокая плотность в пределах 180 дней связана с тем, что за аналогичное время модель способна разместить большее количество работ при меньшем горизонте планирования.

Проверим зависимость времени перепланирования при 20 типах работ (рис. 22 и рис. 23):

Рисунок 22 – Зависимость времени планирования от количества работ при 20 типах работ

Рисунок 23 – Зависимость времени планирования от количества работ при 20 типах работ до момента подскока времени планирования

Проверим, сохранится ли вид графика плотности планирования при большем количестве типов работ. Исходя из результатов, полученных в

экспериментах выше, график должен сохранить свою форму, а средняя плотность должна быть примерно в два раза больше.

График плотности от 20 типов работ представлен на рисунке 24:

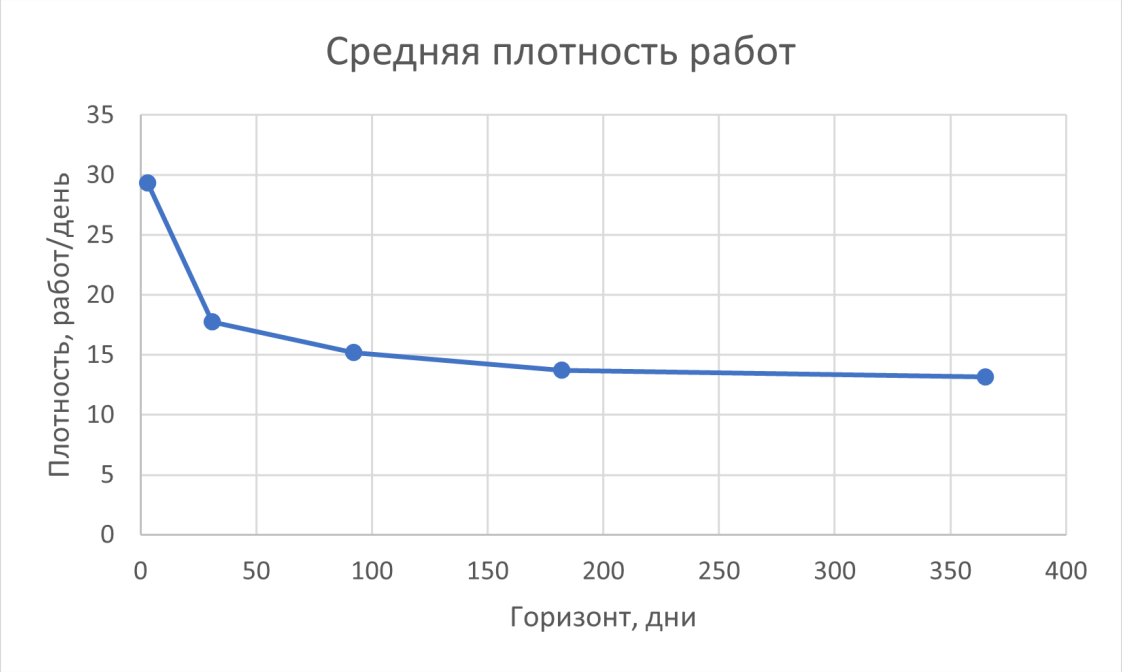

Рисунок 24 – График зависимости средней плотности планирования работ от горизонта планирования

Как видно из рисунка 24, форма графика соответствует форме графика плотности от 10 типов работ, а средняя плотность при долгосрочном планировании увеличилась в 2.4 раза, что так же соответствует ожиданиям.

Последней метрикой для рассмотрения выберем зависимость количества перепланированных работ от количества доступных рабочих. Аналогичный эксперимент с типами работ показал явную зависимость от изменения количества типов, поведение модель в данном эксперименте ожидается похожим.

На рисунке 25 представлены графики зависимости количества работ от горизонта планирования при разном количестве доступных рабочих:

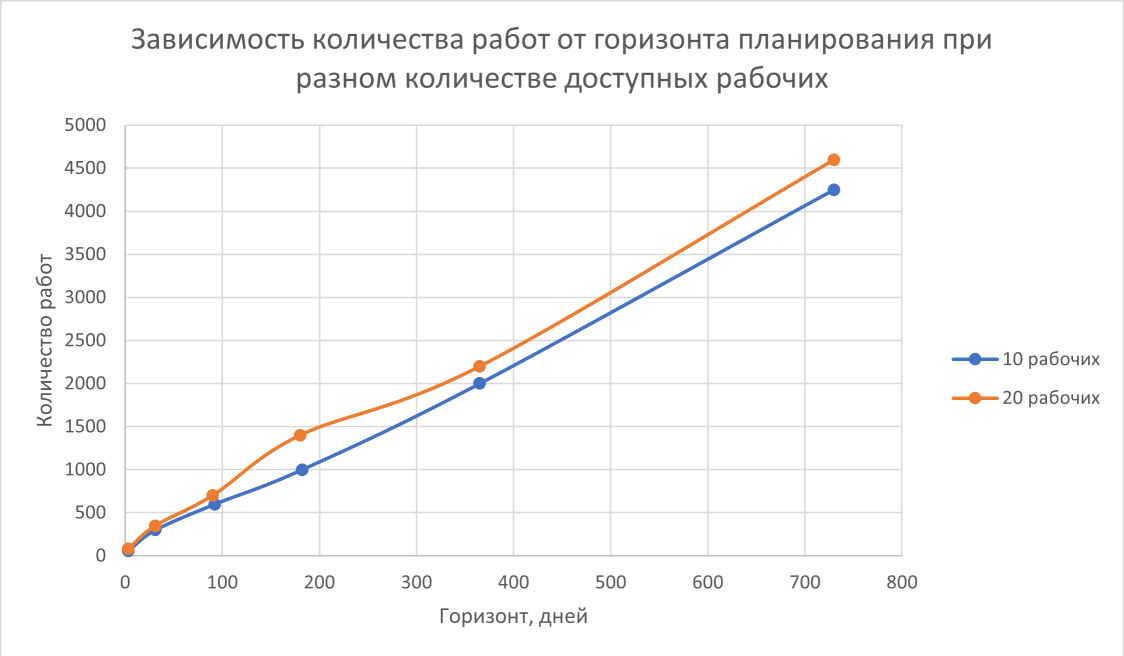

Рисунок 25 – График зависимости средней плотности планирования работ от горизонта планирования

Подобно закону Густафсона-Амдала, в данном случае больший интерес вызывает не быстродействие при определенном объеме вычислений, а объем вычислений при сходном времени. Как видим из рисунка 25, в отличие от изменения количества типов работ, изменение количества доступных рабочих не так сильно повлияло на объем модели при сходном времени. Такое поведение модели можно объяснить тем, что на типы работ накладывается больше ограничений, нежели чем на рабочих, соответственно при ослаблении начальных условий (например, увеличении горизонта планирования или увеличения числа типов работ) становится возможным распределить большее количество работ. Так же необходимо учесть, что рабочие в иерархии инициализации находятся под типами работ, соответственно пласт решений связанных с рабочими частично отсекается сам по себе при нахождении решений для типов работ.

3.2.3 Выводы по итогам эксперимента

По итогам эксперимента было выявлено, что разработанная модель работает в соответствии с поставленной задачей.

Модель корректно отработала в граничных условиях, показав ожидаемы и соответствующие поставленной задаче результаты.

Ключевым параметром системы при работе сервиса планирования оказалось количество ядер процессора, способных параллельно решать задачу перепланирования, так как каждое из ядер берет на себя часть работы по поиску удовлетворяющих решений из всего множества.

Горизонт планирования сильно влияет как на время планирования, так и на количество работ, так как является одним из главных ограничителей для возможного количества распределенных работ.

Количество типов работ значительно влияет на скорость перепланирования и на объем возможных работ за определенное время, так как для типов работ задано больше всего ограничений.

Количество же рабочих меньше всего влияет на процесс перепланирования, так как для них задается меньше ограничений, а часть из множества решений связанных с рабочими отсекается при отсечении решений для типов работ.

Избежать излишнего времени, которое может быть затрачено на перепланирование не удалось, но длительное время планирования является стандартной практикой в области операционных исследований, поэтому лишь частично является минусом в данной ситуации, так как относительно большой объем данных так же возможно перепланировать и в короткие сроки, то есть в пределах нескольких минут.

В целом прототип системы можно назвать пригодным для использования, так как он справляется с представленной задачей.

В дальнейшем планируется наращивание прототипа, а именно:

- Оптимизация инициализации модели и накладываемых ограничений.
- Возможность привязки дополнительных типов сущностей к работам.
- Возможность использования одновременно нескольких сущностей одного типа.
- Доработка схемы базы данных в соответствии с изменениями модели.
- Добавление поддержки составных работ.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была разработана система, предназначенная для ресурсного планирования в системах управления физическими активами на основе открытых технологий.

Актуальность проведенной работы по разработке системы обосновывается отсутствием подобного функционала в присутствующих на рынке ERP и EAM-системах, таких как 1С-ERP, ”TRIM”, ”Галактика ERP” и так далее, а так же заинтересованностью рынка EAM-систем в подобных сервисах.

Проведен анализ доступных методов для решения вопроса ресурсного планирования:

– Мультиагентные системы в целом являются достаточно перспективными в плане использования в ресурсном планировании, так как теоретически способны решить задачу самоорганизации активов предприятия, но данный подход не был широко протестирован в реальных условиях, а сами разработки в этой области, увы, в следствие современных подготов к программированию были фактически заброшены в конце нулевых;
– Комбинаторная оптимизация же представляется наиболее подходящим методом решения поставленной задачи на текущий момент. Ресурсное планирование методами комбинаторной оптимизация не представлено в популярных пакетах EAM и ERP-систем, но явно предоставляет набор устоявшихся практик и решений, показавших себя во множестве проектов сходной тематики.

Был проведен анализ доступных средств в рамках комбинаторной оптимизации, в котором лучшим образом показал себя фреймворк Google OR-Tools. Данный фреймворк не уступает по качеству и скорости решений как свободным аналогам, так и проприетарным конкурентам, является свободным и открытым, а так же может использоваться и для коммерческой разработки.

Были сформированы требования к системе ресурсного планирования на базе комбинаторной оптимизации.

Были разработаны архитектура системы и программа сервиса планирования работ, а так же прототипа EAM-системы для взаимодействия пользователя с сервисом перепланирования.

Система была создана в рамках экосистемы .Net с использованием фреймворка Blazor Server в качестве основного связующего элемента между сервисом, базой данных и непосредственно прототипом EAM-системы.

Построенная система состоит из открытых и свободных элементов платформы .Net Core, а в качестве СУБД использовался свободный и открытый PostgreSQL, поэтому результирующая система так же является полностью открытой и мультиплатформенной.

После создания прототипа системы ресурсного планирования были проведены экспериментальные исследования, в ходе которых было установлено, что система работоспособна и соответствует поставленной задаче.

Научно-техническая новизна работы состоит в разработке и исследовании применимости модели, основанной на принципах комбинаторной оптимизации для реальных задач в области управления активами и ресурсного планирования.

Практическое применение системы ресурсного планирования заключается в возможности непосредственного её использования в ERP и EAM-системах как некий внутренний инструмент, так и как сторонний подключаемый сервис, работающий с данными из ERP.

За период обучения была опубликована статья в журнале ”Инновации. Наука. Образование” на тему ”Операционные исследования на основе различных подходов”.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Видеозапись Intel Fab Tour, [Электронный ресурс], URL: https://www.youtube.com/watch?v=2ehSCWoaOqQ (дата обращения 13.04.2022).

2. Virginia Dignum, Frank Dignum, ”Agents are Dead. Long live Agents!”, UmeåUniversity International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC AAMAS 2020, May 9–13, Auckland, New Zealand

3. Frank Dignum, Jeff Bradshaw, Barry Silverman, Willem van Doesburg, ”Agents for Games and Simulations”, Изд-во Springer, 2009. – 247с.

4. Городецкий В.И., Скобелев П.О, ”Многоагентные технологии для индустриальных приложений: реальность и перспектива”, Труды СПИИ-РАН. 2017. № 55 (6). С. 11-45.

5. Сайт Wikipedia - Исследование операций [Электронный ресурс], URL: https://ru.wikipedia.org/wiki/Исследование_операций (дата обращения 23.02.2022).

6. Сайт фреймворк GLPK [Электронный ресурс], URL: https://www.gnu.org/software/glpk/ (дата обращения 13.03.2022).

7. Сайт LP_Solve [Электронный ресурс], URL: http://lpsolve.sourceforge.net/ (дата обращения 13.03.2022).

8. Сайт minizinc [Электронный ресурс], URL: https://www.minizinc.org/ (дата обращения 13.03.2022).

9. Сайт OR-Tools [Электронный ресурс], URL: https://developers.google.com/optimization (дата обращения 13.03.2022).

10. Сайт CHOCO [Электронный ресурс], URL: https://choco-solver.org/ (дата обращения 13.03.2022).

11. Сайт IBM CPLEX [Электронный ресурс], URL: https://www.ibm.com/analytics/cplex-optimizer (дата обращения 13.03.2022).

12. Сайт GUROBI [Электронный ресурс], URL: https://www.gurobi.com/ (дата обращения 13.03.2022).

13. Giacomo Da Col, Erich C. Teppan, ”Google vs IBM: A Constraint Solving Challenge on the Job-Shop Scheduling Problem”, Electronic Proceedings in Theoretical Computer Science. 2019. С. 259-265.

14. Hvattum, Lars Magnus & Løkketangen, Arne & Glover, Fred, Comparisons of Commercial MIP Solvers and an Adaptive Memory (TabuSearch) Procedure for a Class of 0–1 Integer Programming Problems, Algorithmic Operations Research Vol.7 (2012) 13–20.

15. Сайт Minizinc challenge [Электронный ресурс], URL: https://www.minizinc.org/challenge.html (дата обращения 11.04.2022).

16. Википедия - Диета Стиглера [Электронный ресурс], URL: https://en.wikipedia.org/wiki/Stigler_diet (дата обращения 03.04.2022).

17. .NET vs. .NET Framework for server apps [Электронный ресурс], URL: https://docs.microsoft.com/en-us/dotnet/standard/choosing-core-framework-server (дата обращения 08.04.2022).

18. Сайт Postgres PRO [Электронный ресурс], https://www.postgrespro.ru/ (дата обращения 08.04.2022).

19. Реестр - СУБД Postgres PRO [Электронный ресурс], https://reestr.digital.gov.ru/reestr/301574/?sphrase_id=1426840 (дата обращения 08.04.2022).

20. https://www.syncfusion.com/blazor-components [Электронный ресурс], https://reestr.digital.gov.ru/reestr/301574/?sphrase_id=1426840 (дата обращения 23.02.2022).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД МОДЕЛИ ПЕРЕПЛАНИРОВАНИЯ

```
1  using Google.OrTools.Sat;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Threading.Tasks;
6
7  namespace PlanServer.Models
8  {
9      public static class Planner
10     {
11         private static IEnumerable<DateTime>
   ↪  EachDay(DateTime from, DateTime thru)
12         {
13             for (var day = from.Date; day.Date <=
   ↪  thru.Date; day = day.AddDays(1))
14                 yield return day;
15         }
16         public static async Task<List<OrToolsJob>>
   ↪  OROptimizeAsync(int horizon, List<int> workers,
   ↪  List<int> workers_job_num, List<int> job_types,
   ↪  List<int> job_type_num, List<OrToolsJob> jobs,
   ↪  DateTime reftime, DateTime lastday)
17         {
18             List<OrToolsJob> result = await Task.Run(()
   ↪  => OROptimize(horizon, workers, workers_job_num,
   ↪  job_types, job_type_num, jobs, reftime, lastday));
19             return result;
```

```
20          }
21          private static List<OrToolsJob> OROptimize(int
    ↪ horizon, List<int> workers, List<int>
    ↪ workers_job_num, List<int> job_types, List<int>
    ↪ job_type_num, List<OrToolsJob> jobs, DateTime
    ↪ reftime, DateTime lastday)
22          {
23              Console.WriteLine("Defining model");
24              CpModel model = new CpModel();
25              List<List<List<IntervalVar>>>
    ↪ intervals_by_worker = new();
26             List<List<List<BoolVar>>> bools_by_worker =
    ↪ new();
27              List<List<List<IntVar>>> start_by_worker =
    ↪ new();
28              List<List<List<IntVar>>> end_by_worker =
    ↪ new();
29              List<List<List<int>>>
    ↪ corresponding_jobtype_worker = new();
30              foreach (var w in workers)
31              {
32                  List<List<IntervalVar>>
    ↪ intervals_by_type = new();
33                  List<List<BoolVar>> bools_by_type =
    ↪ new();
34                  List<List<IntVar>> start_by_type =
    ↪ new();
35                 List<List<IntVar>> end_by_type = new();
36                  List<List<int>>
    ↪ corresponding_jobtype_type = new();
37                  foreach (var jt in job_types)
```

```
38                  {
39                      List<IntervalVar>
    ↪ intervals_by_job_by_type = new();
40                      List<BoolVar> bools_by_job_by_type
    ↪ = new();
41                     List<IntVar> start_by_job_by_type =
    ↪ new();
42                      List<IntVar> end_by_job_by_type =
    ↪ new();
43                     List<int> corresponding_jobtype_job
    ↪ = new();
44                      foreach (var job in jobs)
45                      {
46                          if (jt == job.Type && w ==
    ↪ job.WorkerID)
47                          {
48                              var performed_at =
    ↪ model.NewBoolVar($"performed by {w}wID {jt}type
    ↪ {job.ID}jID");
49                              var start = model.NewIntVar(0,
    ↪ horizon, $"start of X {jt} {job.ID}");
50                              var end = model.NewIntVar(0,
    ↪ horizon, $"end of {w} {jt} {job.ID}");
51                              var optint =
    ↪ model.NewOptionalIntervalVar(start,
    ↪ job.TimeSpanMinutes, end, performed_at, $"interval
    ↪ of {w} {jt} {job.ID}");
52                              model.Add(start < end);
53
    ↪ bools_by_job_by_type.Add(performed_at);
```

```
54
55      intervals_by_job_by_type.Add(optint);
56
57      start_by_job_by_type.Add(start);
58                              end_by_job_by_type.Add(end);
59
60      corresponding_jobtype_job.Add(job.ID);
61                          }
62                      }
63
64      intervals_by_type.Add(intervals_by_job_by_type);
65
66      bools_by_type.Add(bools_by_job_by_type);
67
68      start_by_type.Add(start_by_job_by_type);
69                      end_by_type.Add(end_by_job_by_type);
70
71      corresponding_jobtype_type.Add(corresponding_jobtype_jo
72                  }
73
74      intervals_by_worker.Add(intervals_by_type);
75                  bools_by_worker.Add(bools_by_type);
76                  start_by_worker.Add(start_by_type);
77                  end_by_worker.Add(end_by_type);
78
79      corresponding_jobtype_worker.Add(corresponding_jobtype_
80
81              }
82              Console.WriteLine("Model defined");
83
84      ////////////////////////////////////////////////////////
```

```
75              ///
76              //работы по рабочему
77              List<List<IntervalVar>>
    ↪ jobIntervals_by_worker = new();
78              foreach (var w in intervals_by_worker)
79              {
80                  //List<IntervalVar> lst = new();
81                  var flatlist = w.SelectMany(i =>
    ↪ i).ToList();
82                  jobIntervals_by_worker.Add(flatlist);
83              }
84
85              //определим нерабочие (ночные интервалы) -
    ↪ допустим с 9 утра до 9 вечера
86              //даты планирования выбираются с 00:00 по
    ↪ 00:00
87              //отсюда каждый день можно без проблем на N
    ↪ смен (зависят от типа работы/рабочего - ?)
88              //0 + 540 => ночь; +720 => день; + 180 =>
    ↪ остатки ночи
89              List<IntervalVar> sleepIntervals = new();
90              int dayCounter = 0;
91
92              foreach (DateTime day in EachDay(reftime,
    ↪ lastday))
93              {
94
95                  int st = dayCounter * 1440;
96
```

```
97                  if ((day.DayOfWeek ==
    ↪ DayOfWeek.Saturday) || (day.DayOfWeek ==
    ↪ DayOfWeek.Sunday))
98                  {
99                      //по выходным не работаем
100                     int fin1 = st + 1440;
101                     var siSS = model.NewIntervalVar(st,
    ↪ 1440, fin1, $"sat/sun {dayCounter}");
102                     sleepIntervals.Add(siSS);
103                 }
104                 else
105                 {
106                     //по будням - с 9 до 9
107                     int fin = st + 540;
108                     var si = model.NewIntervalVar(st,
    ↪ 540, fin, $"sleep 1 of day {dayCounter}");
109                     sleepIntervals.Add(si);
110                     st = fin + 720; //пропускаем
    ↪ рабочие 12 часов
111                     fin = st + 180;
112                     //второй ночной интервал
113                     var si2 = model.NewIntervalVar(st,
    ↪ 180, fin, $"sleep 2 of day {dayCounter}");
114                     sleepIntervals.Add(si2);
115                 }
116
117                 dayCounter++;
118             }
119
120
121             //работы одного рабочего не пересекаются
```

```
122             foreach (var ji in jobIntervals_by_worker)
123             {
124                 ji.AddRange(sleepIntervals);
125                 model.AddNoOverlap(ji);
126             }
127 
128 
129             ////////////////////////////////////////
130             ///разделяем работы по типу работы
131             ///
132             List<List<IntervalVar>>
    ↪ jobIntervals_by_type = new();
133 
134             foreach (var t in job_types)
135             {
136                 List<IntervalVar> jlst = new();
137                 jobIntervals_by_type.Add(jlst);
138             }
139 
140             foreach (var i_worker in
    ↪ intervals_by_worker)
141             {
142                 for (int i_type = 0; i_type <
    ↪ i_worker.Count; i_type++)
143                 {
144                     var jflatlist = i_worker[i_type];
145 
    ↪ jobIntervals_by_type[i_type].AddRange(jflatlist);
146                 }
147 
148             }
```

```
149
150             //для каждого типа работы - все работы не
    ↪ пересекаются
151             foreach (var ji in jobIntervals_by_type)
152             {
153
154                 model.AddNoOverlap(ji);
155             }
156
157
    ↪ ////////////////////////////////////////////
158             ///разделяем performed_at по типам работы
159             Console.WriteLine("разделяем performed_at
    ↪ по типам работы");
160             List<List<BoolVar>> performedAt_by_type =
    ↪ new();
161
162             foreach (var t in job_types)
163             {
164                 List<BoolVar> perflist = new();
165                 performedAt_by_type.Add(perflist);
166             }
167
168             foreach (var b_worker in bools_by_worker)
169             {
170                 for (int b_type = 0; b_type <
    ↪ b_worker.Count; b_type++)
171                 {
172                     var flatperf = b_worker[b_type];
173
    ↪ performedAt_by_type[b_type].AddRange(flatperf);
```

```
174                 }
175             }
176             for (int i = 0; i <
    ↪ performedAt_by_type.Count; i++)
177             {
178 
    ↪ model.Add(LinearExpr.Sum(performedAt_by_type[i])
    ↪ <= job_type_num[i]);
179             }
180             ////////////////////////////////////
181             ///разделяем performed_at по рабочим
182             List<List<BoolVar>> performedAt_by_worker =
    ↪ new();
183 
184             foreach (var t in bools_by_worker)
185             {
186                 //List<IntVar> worklist = new();
187                 var flatwork = t.SelectMany(i =>
    ↪ i).ToList(); ;
188                 performedAt_by_worker.Add(flatwork); ;
189             }
190 
191 
192             //для каждого рабочего колво работ
    ↪ запланированных <= необходимому
193             for (int i = 0; i <
    ↪ performedAt_by_worker.Count; i++)
194             {
195 
    ↪ model.Add(LinearExpr.Sum(performedAt_by_worker[i])
    ↪ <= workers_job_num[i]);
```

```
196             }
197
198             ////////////////////////////
199             ///
200             //разделим старты и концы работ по типам
201             List<List<IntVar>> startAt_by_type = new();
202
203             foreach (var t in job_types)
204             {
205                 List<IntVar> perflist = new();
206                 startAt_by_type.Add(perflist);
207             }
208
209             foreach (var s_worker in start_by_worker)
210             {
211                 for (int s_type = 0; s_type <
    ↪  s_worker.Count; s_type++)
212                 {
213                     var flatstart = s_worker[s_type];
214
    ↪  startAt_by_type[s_type].AddRange(flatstart);
215                 }
216             }
217
218             List<List<IntVar>> endAt_by_type = new();
219
220             foreach (var t in job_types)
221             {
222                 List<IntVar> perflist = new();
223                 endAt_by_type.Add(perflist);
224             }
```

```
225
226             foreach (var s_worker in end_by_worker)
227             {
228                 for (int s_type = 0; s_type <
    ↪ s_worker.Count; s_type++)
229                 {
230                     var flatend = s_worker[s_type];
231
    ↪ endAt_by_type[s_type].AddRange(flatend);
232                 }
233             }
234
235             //соблюдай-ка порядок работ
236             //проходим все типы работ что попались
237             for (int k = 0; k < startAt_by_type.Count;
    ↪ k++)
238             {
239                 string varname =
    ↪ startAt_by_type[k][0].Name();
240                 varname = varname.Substring(11, 1);
241                 //если тип третий
242                 if (Int32.Parse(varname) == 3)
243                 {
244                     //foreach (var s in
    ↪ startAt_by_type[k])
245                     //{
246
247                     //}
248                     for (int i = 0; i <
    ↪ startAt_by_type[k].Count - 1; i++)
249                     {
```

```
250                         model.Add(startAt_by_type[k][i
    ↪ + 1] > endAt_by_type[k][i]);
251                     }
252                 }
253             }
254 
255             //для третьего типа(плановая работа) -
    ↪ работы не чаще раза в день
256             for (int k = 0; k < startAt_by_type.Count;
    ↪ k++)
257             {
258                 string varname =
    ↪ startAt_by_type[k][0].Name();
259                 string secondvarname = varname;
260                 varname = varname.Substring(11, 1);
261                 //если тип третий
262                 if (varname == " ")
263                 {
264                     varname =
    ↪ secondvarname.Substring(11, 2);
265                 }
266                 if (Int32.Parse(varname) == 3 ||
    ↪ Int32.Parse(varname) == 13)
267                 {
268                     for (int i = 0; i <
    ↪ startAt_by_type[k].Count - 1; i++)
269                     {
270                         model.Add(startAt_by_type[k][i
    ↪ + 1] - endAt_by_type[k][i] >=
    ↪ 1440).OnlyEnforceIf(performedAt_by_type[k][i]);
```

```
271                         //model.Add(startAt_by_type[k][i
    ↪  + 1] > endAt_by_type[k][i]);
272                         }
273                     }
274                 }
275
276
277
278                 ////////////////////////////
279
280             var flatbools = bools_by_worker.SelectMany(i
    ↪  => i).SelectMany(i => i);
281                 Console.WriteLine("Starting Optimization
    ↪  at:");
282
    ↪  Console.WriteLine(DateTime.Now.ToString("MM/dd/yyyy
    ↪  HH:mm:ss"));
283                 //целевая функция - максимизировать
    ↪  количество работ
284                 model.Maximize(LinearExpr.Sum(flatbools));
285                 //Console.WriteLine("test");
286                 var solver = new CpSolver();
287                 solver.StringParameters =
    ↪  "num_search_workers:1";
288                 var status = solver.Solve(model);
289
290
    ↪  Console.WriteLine("##########################");
291                 Console.WriteLine("Всего работ:" +
    ↪  jobs.Count.ToString());
```

```
292             Console.WriteLine("Распределено работ: " +
    ↪ solver.ObjectiveValue);
293 
    ↪ Console.WriteLine("##########################");
294 
295             List<OrToolsJob> PlannedJobs = new();
296 
297             //принтим если есть решение
298             if (status == CpSolverStatus.Feasible ||
    ↪ status == CpSolverStatus.Optimal || status ==
    ↪ CpSolverStatus.Unknown)
299             {
300                 for (int w = 0; w < workers.Count; w++)
301                 {
302 
303                     for (int jt = 0; jt <
    ↪ job_types.Count; jt++)
304                     {
305 
306                         for (int job = 0; job <
    ↪ bools_by_worker[w][jt].Count; job++)
307                         {
308 
    ↪ //Console.WriteLine(bools_by_hour[i][j][k].Count());
309 
    ↪ //Console.WriteLine("----------");
310                             if
    ↪ (solver.Value(bools_by_worker[w][jt][job]) == 1)
311                             {
```

```
312                                     PlannedJobs.Add(new
    ↪ OrToolsJob(corresponding_jobtype_worker[w][jt][job],
    ↪ job_types[jt],
    ↪ (int)solver.Value(start_by_worker[w][jt][job]),
    ↪ (int)solver.Value(end_by_worker[w][jt][job]),
    ↪ reftime, workers[w]));
313                                     //для каждого воркера,
    ↪ каждый джобтайп, каждая работа конкретного типа -
    ↪ начало + конец
314                                 }
315                             }
316                         }
317
318                     }
319
320                 }
321                 Console.WriteLine("Finished at:");
322
    ↪ Console.WriteLine(DateTime.Now.ToString("MM/dd/yyyy
    ↪ HH:mm:ss"));
323                 return PlannedJobs;
324             }
325         }
326 }
```