Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки Прикладная математика и информатика

РАБОТА ПРОВЕРЕНА
Рецензент,

_____
«____»_____2018г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н., доцент
_____А.А.Замышляева
«____»_____2018 г.

Составление расписания при случайном времени выполнения работ с учётом отказа оборудования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–01.03.02.2018.42.ПЗ ВКР

Руководитель работы, доцент
_____/Е.Ю.Алексеева
«____»_____2018 г.
Автор работы
студент группы ЕТ-412
_____/Н.В.Бондаренко
«____»_____2018 г.
Нормоконтролер, к.э.н., доцент
_____/Д.А.Дрозин
«____»_____2018 г.

Челябинск 2018

АННОТАЦИЯ

Бондаренко Н.В Составление расписания при случайном времени выполнения работ с учетом отказа оборудования. – Челябинск: ЮУрГУ, ЕТ-412, 48с., 11 ил., библиогр. список – 30 наим., 1 прил.

В данной работе рассмотрены классификации систем составления расписания, классификации задач теории расписания и типы алгоритмов.

Составлена математическая модель системы составления расписания и модифицирован алгоритм. Реализован графический интерфейс пользователя. Разработано и отлажено приложение для составления расписания при случайном времени выполнения работ с учетом отказа оборудования. Проведена оценка качества получаемых расписаний на примере модельных условий.

Программа реализована на языке программирования C#. В приложении приведен текст программы.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

С расширением бизнеса и увеличением конкуренции на рынке, большая часть малых и средних предприятий испытывает необходимость в повышение эффективности производства, снижение затрат и себестоимости за продукцию. Малое предприятие, работая в условиях некоторых ограничений, имеет возможность успешно функционировать и конкурировать в данной отрасли в основном за счет своих резервов, выявить которые можно за счет математических моделей и алгоритмов составления расписаний и прогнозирование производства. Повышение эффективности производства достигается за счет внедрения системы оперативного управления, составление расписания.

Во всем мире собрано большое количество математических моделей и методов решения задач составления расписания. Значительная часть результатов показывает сложность проблемы. Как показали исследования, применение в реальных обслуживающих системах точных алгоритмов и даже приближенных алгоритмов с достаточно хорошими априорными оценками не представляется возможным [1]. Делаем вывод, что работу по реализации математических моделей составления оптимальных расписаний в конкретных условиях не следует считать завершенной.

В рамках теории расписания изучаются главные математические вопросы построения анализа и моделей оптимизации расписания. В известной работе Генри Гантта 1903г [2] было впервые рассмотрено данное научное направление, он предложил, что на данный момент носит название диаграмма Гантта. Кроме того, проблемами теории расписания занималось еще множество ученых, среди них Р.В. Конвей, Г. Вагнер [3], В.Л. Максвелл [4], Ю.Н Сотсков, Т.П. Подчасова [5] и другие.

Модели изучаемые в теории расписания показывают нестандартные ситуации, появляющиеся при календарном планировании различный видов человеческой деятельности. Большое количество моделей и их универсальность постепенно увеличивается, охватывая все более широкую сферу производственных расписаний, информационно-вычислительных процессов и т.д.

Составление производственных расписаний являются NP-трудными задачами с большой размерностью и ограничениями сложной формы. Изучив их можно сделать вывод, что часть задач своеобразны, а другая часть сложно формализуемы. Для возможности решения таких задач выведены классы алгоритмов, основными из которых являются: приближённые, эвристические, методы сокращения перебора. Для получения расписания, которое будет отвечать поставленным требованиям, применяются эвристические алгоритмы, проведение оптимизации для которых можно избежать.

В теории расписания базовым понятием является понятие работы – элементарная планируемая часть процесса. При составлении плана выполнения работ, в нем фиксируется объем работ и моменты начала и окончания каждой

работы. Компоненты деятельности, такие как оборудование, энергия, материалы, исполнители входят в понятие ресурса работ.

На сегодня системы автоматизированного составление производственных расписаний (MES, APS) отсутствуют на большинстве малых предприятий, лидирующих в инновационных разработках. Это связано со сложностью внедрения систем и огромной стоимостью, не большим количеством персонала, отсутствие налаженного документооборота, обобщенной структурой производства и развитой технологической базы. В данных условиях возникает необходимость в разработке алгоритмов и моделей составления производственных расписаний, которые не требуют больших вложений при эксплуатации и внедрении, простых и гибких в применении, предусматривающих специфику производства.

Классическими критериями задач теории расписаний являются: минимизация суммарного времени окончания обслуживания, минимизация суммарного запаздывания, минимизация числа запаздывающих требований, минимизация максимального времени окончания обслуживания (задача на быстродействие).

В работе Грэхэма была введена классификация для задач теории расписания. По этой классификации каждой задаче соответствует , где α обозначает тип и количество доступных приборов, β описывает дополнительные ограничения задачи, γ определяет критерий задачи.

Возникающие на практике задачи постоянно совершенствуют организацию процесса построения расписания и математические методы. Огромный интерес представляет разработка методов, дозволяющих решать задачи составления расписания на основе эвристик. В большинстве случаев разработка таких систем осложняется необходимостью решения задачи за допустимое время. Такие задачи постоянно возникают на производствах любой сложности. Цель данной работы моделирование системы составления расписания с учетом отказа оборудования.

Чтобы реализовать эту задачу выбран подход, который комбинирует различные алгоритмы поставленной задачи. Это позволит формировать решения за приемлемое время, с хорошей точностью которые соответствуют адекватным требованиям. Часть ограничений не имеет принципиального характера и при построении модели либо упрощается, либо не учитывается. Результатом работы является расписание, близкое к оптимальному, это решить поставленную задачу с учетом всех ограничений и получит требуемый результат.

1 ТЕОРИЯ РАСПИСАНИЙ И СИСТЕМ ИХ СОСТАВЛЕНИЯ

1.1 Современное состояние проблемы составлений расписаний

1.1.1 Классификация систем составлений расписаний

Задача составления расписания на производстве включает в себя множество понятий. Данную задачу разбивают на множество классов подзадач, и для каждой класса применяют собственные алгоритмы и методы.

Задача составления расписания выпуска продукции предприятием является самостоятельным направлением автоматизации. Совмещение с разными системами управления производства зависит от получения оперативной исходной информации и использованием полученных данных в других подсистемах.

В текущем этапе развития автоматизационных систем, задачи управления производством, в том числе задачи составления расписания выпуска продукции предприятием выведены в отдельный класс. MES (Manufacturing Execution System) – это система управления производством, которая связывает воедино все бизнес-процессы предприятия с производственными процессами, оперативно поставляет объективную и подробную информацию. Кроме того, система MES проводит анализ и определяет наиболее эффективное решение проблемы [6].

В некоммерческой ассоциации MESA (Manufacturing Enterprise Solutions Association), определение MES является общепринятым, MESA также определяет производителей и внедрение MES систем. MES – это автоматизированная система управления производственной деятельностью предприятия, которая в режиме реального времени:

– контролирует;

– документирует производственные процессы от начала формирования заказа до выпуска готовой продукции;

– планирует;

– оптимизирует.

Ассоциация MESA определила 11 основных функций MES:

– Диспетчеризация производства (DPU);

– Управление документами (DOC);

– Контроль состояния и распределение ресурсов (RAS);

– Оперативное/Детальное планирование (ODS);

– Управление качеством продукции (QM);

– Управление персоналом (LM);

– Сбор и хранение данных (DCA);

– Управление производственными процессами (PM);

– Управление производственными фондами (техобслуживание) (MM);

– Анализ производительности (PA);

– Отслеживание истории продукта (PTG).

В данный момент функции составления расписания уделяется повышенное внимание [7]. Дело в том, что не известна логика решения оптимизационной задачи построения детального расписания работы персонала и оборудования. Данная задача не тривиальна. Часть других ключевых функций понятный по логическому содержанию, но сложны в реализации. Основными назначениями задач MES будут управление и контроль статусами трех основных составляющих системы автоматизации предприятия:

– EAM – как контроль статусов оборудования и его загрузка в ходе выполнения плана;

– АСУТП – как контроль за ходом плана, его текущим состоянием;

– HRM – как контроль статусов персонала и его загрузка в процессе обслуживания хода технологии и технологического оборудования [8].

Эти задачи являются первичной информацией для системы управления выполнением производственных заданий, под которыми можно понять поддержание хода производственного процесса как единой системы взаимодействия всех компонентов: действий персонала и средств производства. Вышеперечисленные данные систем MES позволят в реальном времени руководить бизнес-процессами производства и обеспечить:

– контроль исполнения ПЗ;

– сохранение истории ПЗ;

– генерирование производственных заданий (ПЗ);

– расчет времени и стоимости исполнения ПЗ;

– ведение статистики выполнения ПЗ;

– планирование материальных, технологических и человеческих ресурсов, необходимых для исполнения ПЗ;

– передачу данных о ПЗ в смежные и вышестоящие системы ERP управления предприятием.

Внедряя системы MES в разные организации встает вопрос о том, как реализовать функцию составления расписания. Есть выбор из 2-х вариантов: составлять расписание «с нуля» или настроить готовое расписание под предприятие. Нужно еще определиться с требованиями к оптимизации: необходима оптимизация или достаточно найти допустимый план. Если необходима оптимизация, то возникает вопрос критерия оптимальности. Более трудоемкая оптимизация по нескольким параметрам. Независимо от выбора важно иметь представление об алгоритмах решения задачи составления расписания.

Интеграция системы составления расписания с другими компонентами MES-системы – это один из важных моментов. Она будет использоваться эффективно, если компоненты используют единую информационную среду и составленное расписание получает данные в реальном времени из виртуального предприятия.

Тогда получается легко реализовать функцию составления расписания по событиям, происходящем в реальном времени.

Решение задачи составления расписания по-разному интерпретируется компаниями, использующими MES системы. Иногда под составление расписания понимается тривиальный пересчет плановых показателей и поддержка документооборота. Если предприятие предполагает трудные запутанные связи между операциями, то составление расписания, в нахождении оптимального расписания может быть не решаема как в вычислительной трудоемкости поиска, так и аналитической постановке алгоритма решения. Под специфику работы типовых предприятий уже давно были реализованы системы с составлением расписания выполняющие ее, в той или иной степени.

У предприятий в контексте из автоматизации различают производства V-типа и A-типа [7]. V-тип – производства с большим количеством сырья и малой готовой продукцией, A-тип – производства с малым количеством сырья, но с большим количеством видов продукции. Например, для V-типа – производства, направленные на сборку машин, агрегатов, а для A-типа – производства в сфере химической промышленности. В производстве V-типа гораздо проще решить задачу составления оптимального расписания, чем в производстве A-типа, т.к. для V-типа все операции сводятся по времени на конечный продукт, а в A-типе большее значение имеет зависимость выхода от используемого сырья и его долевой состав.

Большая часть задач теории расписания NP-полные, и зачастую их алгоритмы решения имеют экспоненциальную сложность. То есть, если есть алгоритм решения конкретной задачи, то его вычислительная трудоемкость, с большой вероятностью будет экспоненциально расти от размерности задачи (количества объектов). Задачи составления расписания на предприятиях, будут иметь довольно большую размерность. Сокращение области поиска оптимального решения, зависит от специфики оборудования. Это отображается в том, что, на производствах возможно сначала выделить цикличность и проводить оптимизацию в рамках одного цикла, а затем решение распространить на весь интервал планирования.

Использование эвристик является еще одним путем снижения вычислительной сложности задачи. Тогда цена решения – отсутствие строгого доказательства, того что найденное решение оптимально, это решение можно назвать только удачным. Однако, существует в этом потребность – необходимо найти удачное решение. Для этого нужно определить пороговое качество (критерий) расписания, при нем оно будет считаться удовлетворительным.

1.1.2 Современные системы составления расписания

Нынешний рынок систем автоматизации управления производством полон и востребован. Количество продуктов многообразно и имеет два главных направления: удобно настраиваемые системы широкого круга использования и

узкоспециализированные системы автоматизации. Внедрение первых могут привести к большим трудовым и финансовым затратам, но также дают колоссальный (при условии грамотного внедрения) эффект. Поэтому внедрение таких продуктов зачастую не оправдано или невозможно в силу ряда причин. А при использовании в процессе автоматизации узкоспециализированных продуктов оправдано с экономической точки зрения. Одним из минусов является отсутствие много охватывающей поддержки разработчика после внедрения. Кроме этого предприятие испытывает минимум трудностей, связанных с кардинальной переделкой системы менеджмента или с изменением учета, такая автоматизация наиболее легкая для предприятия. В этой главе приведем небольшой обзор систем широкого профиля отечественных и иностранных производителей, рассмотрим программный продукт 1С Предприятие 8.3, в котором уже добавлена часть функций, реализованных в рамках данного проекта, и который является продолжение программы автоматизации производства после 1С Предприятие 7.7.

У иностранных производителей систем автоматизации производства, есть система SIMATIC IT от SIEMENS [9,10]. Она выполняет комплексное моделирование производственных процессов, определяет их возможности и получает данные с ERP уровня и уровня производства в реальном масштабе времени. Система SIMATIC IT предоставляет целый ряд преимуществ. Во-первых, моделировать можно структуры производства и сложные деловые процессы, которые можно объединить эффективным способом. Во-вторых, процессы моделирования остаются понятными и прозрачными, и независимыми от функционирования реальных систем управления. В-третьих, в любой точке предприятия можно произвести моделирование и все процессы могут быть стандартизированы и самые удачные методы управления используются в масштабах всего предприятия. Система SIMATIC IT обеспечивает переход от результатов моделирования к выполнению принятых решений и управлению новыми приложениями MES. В системе SIMATIC IT созданные модели производства, могут сохраняться в библиотеках и загружаться в любое время в другие проекты. Это существенно сокращает затраты на дальнейшее проектирование.

SIMATIC IT полностью отвечает требованиям стандарта ISA-S95 к MES системам.

Основным компонентом является SIMATIC IT Production Suite, который несет в себе модель производства с учетом вспомогательных потоков информации. С помощью Production Suite можно решить такие задачи, как:

- учет работающего персонала;
- учет материалов;
- управление заказами;
- построение всей модели производства;
- организация обмена данными в MES системе;

– учет времени простоя и работы оборудования;
– контроль хода процесса и многие другие.

Кроме этого с этой системой могут независимо работать еще три компонента.

SIMATIC IT Historian выполняет хранение и обработку всех данных, получаемых системой MES из разных источников [9]. Такими данными могут быть данные из сторонних баз данных, из других приложений системы, данные с уровня АСУ ТП. Эти данные при необходимости проходят статическую или математическую обработку, или хранится в долговременном архиве и предоставляться пользователю в различном виде. Вместе с компонентом Production Modeler SIMATIC IT Historian предоставляет инструмент для расчета технико-экономических показателей (ТЭП) или Key Performance Indicator (KPI), а также осуществлять контроль работы оборудования и времени простоя.

SIMATIC IT Unilab реализует систему поддержки лабораторных исследований.

SIMATIC IT Interspec обеспечивает поддержку спецификаций продукта на протяжении его работы.

В целом SIMATIC IT представляет собой конструктор, с помощью которого можно собрать MES систему для конкретного производства. Большие возможности компонентов с простым механизмом их соединения явной модели производства и средства подключения сторонних приложений – это позволяет создать мощную систему, подходящую текущему процессу и легко редактируемую в случае новых требований.

1.2 Оперативное составление расписания

1.2.1 Введение в оперативное управление

Говоря про оперативное управление будем рассматривать малую часть задач. Нас интересует понятие составление расписания выпуска продукции, которое включается в понятие оперативного управление.

Задача теории расписания считается заданной, если определены:
– порядок прохождения машин;
– критерии оценки расписаний [5];
– подлежащие выполнению операции и работы;
– типы и количество машин, выполняющих операцию.

В зависимости от характера полученной работы различают два вида задач: динамические и статические. В динамических задачах работа поступает в систему в разные моменты времени, которые можно предсказать в статическом смысле. Поэтому сложно определить моменты будущих поступлений. В статических задачах в свободную систему поступает определенное количество работ одновременно. Упорядочение в статических и динамических задачах требует разных методов решения.

В каком порядке будет выполнена машинами одна работа определяет, является ли система машин:

– системой произвольного типа;

– со случайным порядком выполнения работ;

– конвейерной.

В системах произвольного типа каждая операция можно выполнить определенной машиной.

В системе со случайным порядком выполнения работ любую операцию может выполнить любая машина, все машины идентичны.

В конвейерной системе последовательность прохождения машин одинакова для каждой из работ. Это означает, что есть такая нумерация машин, что для одинаковых работ номер машины, выполняющей операцию A, меньше номера машины, выполняющей операцию B, если B впереди A.

В общем смысле задача составления расписания можно сформулировать, как задача организации во времени работ на приборах [11]. Множество работ (требований) обозначим как , станки (приборы) . Считаем, что каждая работа обслуживается для обработки детали . Предполагается, что в любой момент времени каждый прибор может выполнить только одну работу, а каждая работа может обслуживаться только на одном приборе. Каждую работу будем характеризовать следующими параметрами:

– приоритет, характеризующий важность работы;

– директивный срок, работу которого нужно обслужить;

– число операций, на которые разбивается работа;

– длительность -ой операции работы .

Чтобы описать конкретную постановку задачи составления расписания будем использовать трехпозиционную формулу [12].

Первая позиция это особенности использования приборов [13].

В частности

, (1.1)

где – случаи одно операционных работ;

– случаи много операционных работ.

: параллельная система станков. Каждая работа может выполняться любым станком , а длительность зависит от станка ее выполняющую.

: один станок, т.е. m=1.

: параллельная система станков одинаковой производительности.

: последовательная система станков выполняет работы , представляющие собой цепи операций , в которых операция должна выполняться на станке с номером в течении времени . Запись означает, что операция не может

выполняться до завершения операции , причем для всех . Таким образом, детали могут проходить обработку по любым назначенным маршрутам, причем любые два соседних станка в одном маршруте обязательно различны.

Если – положительное целое, то , т.е. количество станков фиксировано. Если , то количество станков произвольно.

Вторая позиция

, (1.2)

где

;

;

определяет особенности выполнения работ. .

: есть s ресурсов в количестве , . Для выполнения работы требуется ресурс каждого типа в количестве .

: есть только один тип ресурсов.

: ограничения ресурсов отсутствуют.

: на множестве работ задан порядок со следующей интерпретацией: сравнение означает, что работа не может начинаться до завершения работы .

: диаграмма Хассе порядка лесом с корневыми деревьями.

: диаграмма Хассе порядка является набором отдельных путей.

: ограничения предшествования работ отсутствуют.

: одинаковые длительности одно операционных работ.

: одинаковые длительности операций всех работ.

: длительности операций равны 1 или 2.

: длительности операций зависят только от производительности станков.

: ограничения на длительность отсутствуют.

Третья позиция показывает критерий эффективности расписания. Любое расписание показывает для каждой работы значения следующих параметров: – момент завершения, – временное смещение, , если и , если – единичный штраф за опоздание к назначенному директивному сроку.

Широко используются следующие критерии: максимальный момент завершения или длина расписания [11]. Расписание, минимизирующее , называется максимальным по быстродействию; минимизация максимального временного смещения; – средний момент завершения;

средний взвешенный момент завершения; суммарное запаздывания; – число опозданий, взвешенное число опозданий.

Несколько примеров постановок задач составления расписания с помощью трехпозиционной формулы.

– задача Джонсона [8]: составить оптимальное по быстродействию расписание двух операционных работ для конвейерной системы с двумя станками с любой по длительности операцией.

– задача Мура [5]: составить расписание одно операционных работ на одном станке с произвольными длительностями, в котором число опоздавших работ было минимальным.

– задача Лившица [5]: минимизация максимального штрафа в задаче с одним станком, где штрафная функция имеет сложный вид, например

, (1.3)

где – критерий важности поставки партии в заданный директивный срок

– единичный штраф за опоздание.

– задача Грэхема [14]: составить оптимальное по быстродействию расписание одно операционных работ с любыми ограничениями для параллельной системы станков одинаковой производительности и с любой длительностью.

Вводя отношения предшествования, можно обобщить данную задачу. Это необходимо делать, например, очередность изготовления некоторых партий деталей нужно строго регламентировать. Введем ограничение предшествования , которое означает что партия должна быть изготовлена раньше партии . Такое обощение приводит к модели – задаче Лорера [5].

1.2.2 Классическая постановка задачи теории расписаний

Рассмотрим классическую постановку задачи теории расписаний для нескольких приборов. Множество требований

, (1.4)

должно быть выполнено без прерываний на одном или нескольких приборах , где , которые могут обслуживать не более одного требования в каждый момент времени [11]. Время обслуживания одного требования на приборе обозначается как . Момент, когда требование *k* становится доступным для обслуживания, задаётся временем поступления . Требование *k* может иметь вес (важность требования) и директивный срок выполнения . Между требованиями можно задать отношения предшествования в виде ацикличного ориентированного графа *G*. В момент окончания обслуживания требования *k* при расписании будем обозначать через . Обозначим как временное смещение требования *k* в расписании

, (1.5)

также определим

(1.6)

как его запаздывание. Обозначение означает как запаздывание требования k в расписании или нет: , если k запаздывает, иначе (требование k обслуживается вовремя). В задачах теории расписания классическими критериями являются: минимизация максимального временного смещения, минимизация максимального времени окончания обслуживания (задача на быстродействие), минимизация суммарного запаздывания, минимизация суммарного времени окончания обслуживания, минимизация числа запаздывающих требований. Через обозначим расписание из элементов множества N, обслуживаемых на приборе , где , множество требований в расписании обозначается

, (1.7)

а через

, (1.8)

где ;

расписание для всего множества требований . Разумеется, рассматриваем только возможные расписания без искусственных простоев приборов, соответствующие отношениям предшествования. Множество допустимых расписаний не пусто, для любого ациклического графа G. Момент окончания обслуживания требования k на приборе при расписании определяется как

, (1.9)

где – согласно графу G, это множество требований предшествующих требованию k;

– требования, согласно расписанию на приборе , предшествующие обслуживанию требования .

Множество расписаний обслуживания требований множества N, допустимых относительно графа предшествования G, будем обозначать через .

1.2.3 Классификация задач

В данной главе рассмотрим задачи, решение которых, в дальнейшем могут быть использованы для решения поставленной задачи. В реальном производственном цикле используется аналог конвейерного производства, где цикл одной работы состоит из 2 или 3 операций. Понятно, что критерием оптимальности поставленной задачи является временной критерий, а операции задают на множестве отношения предшествования. Поэтому из большого

количества задач в теории расписаний рассмотрим лишь пару, решение которых, приведет к решению поставленной задачи.

Рассмотрим задачу Лорера [5] . Эту задачу можно решить с помощью упорядочения работ по неубыванию директивных сроков. Критерием эффективности расписания оценивается штрафной функцией

, (1.10)

где – единичный штраф за опоздание;

– критерий важности поставки партии в заданный директивный срок

Алгоритм решения данной задачи строит оптимальную очередность работ (партия это результат выполнения работы), начиная с последней [5]. Тогда первая работа начинается в нулевой момент времени. И при любой очередности последняя работа завершится в момент времени . Последнюю работу следует выбрать из условия

, (1.11)

где k – выбирается по таким работам, для которых максимальны по порядку.

Для выбора предпоследней работы нужно исключить из множества работ и проделать ту же процедуру с множеством оставшихся работ.

Из условия (11) вытекает очевидное свойство оптимальной очередности :

, (1.12)

где момент завершения работы , .

Рассмотрим, что описанный алгоритм строит оптимальную очередность работ. Обозначим значение критерия для очередности и предположим противное: существует очередность , для которой

. (1.13)

Найдем в работу из условия

, (1.14)

т.е. – это первая работа, которая не совпадает с . Такая работа обязательно найдется, иначе . Заметим, что в силу (1.14) и для некоторого . Построим из очередность , перемещая работу в j-ю позицию, т.е. . Работа в и работа в и завершается в один и тот же момент , причем , поэтому из (12) вытекает .

Выходит, при переходе от к штраф: в позиции *j* не увеличивается в силу последнего неравенства; в позициях тоже не увеличивается, т.к. штрафные функции неубывающие, а в моменты завершения находящихся в этих

позициях работ соответственно уменьшились на ; в позициях – остался прежним, поскольку моменты завершения работ остались прежними. Следовательно, .

К применим ту же процедуру и получим очередность , затем таким же образом – и далее . Последнее равенство обязательно выполнится при некотором , поскольку большим номерам будут соответствовать очередности с более длинными окончаниями последовательности, совпадающими с окончаниями последовательностей той же длины в . Последовательности будет соответствовать цепь неравенств , которая противоречит неравенству (13), что и доказывает оптимальность .

Еще один классом задач теории расписаний являются задачи составления мультипроцессорного расписания. В общем случае выполнение работ не должно прерываться, каждая работа выполняется любым прибором, а каждый прибор выполняет одну работу.

Задача известна под названием «задача о разбиении» [15] или «задача о камнях» и является NP-трудной. Чтобы сделать оптимальные по быстродействию расписания удается составить за небольшое время лишь с помощью приближенных полиномиальных алгоритмов.

Самым распространенными из них является так называемые списочные алгоритмы, сделанные на списочном методе составления расписаний. Этот метод может применяться в самых общих ситуациях и использует наиболее лучшие идеи экономии времени.

Рассмотрим задачу [11]. Как и всякий другой списочный алгоритм, в этом случае он реализуется в два этапа.

На первом этапе готовится список приоритетов , в котором работы упорядочиваются по убыванию значений какого-либо параметра работ , который обозначим параметром приоритетов. Местоположение работы в списке определяет ее ценность: чем раньше она встречается в списке, тем выше ее ценность. Рассмотрим следующие параметры приоритетов: ДЛ – длительность; КП – длина критического пути, другими словами максимальная сумма длительностей работ, которые лежат вдоль максимальной (по порядку предшествования) цепи, исходящей из данной работы; –ДЛ – минус-длительность.

На втором этапе работы составляются по списку приоритетов на приборы по мере их освобождения. Это осуществляется следующим образом. Пусть некоторые работы уже включены в расписание и вычеркнуты из списка . После выполнения этих работ станки освобождаются в моменты времени . Выберем станок , который освобождается не позже остальных станков, т.е. . Если таких станков несколько, то среди них нужно выбрать станок с минимальным номером . Затем оставшийся список

просматривается слева направо и находится первая по списку работа , готовая к выполнению не позже остальных готовых работ списка. Работа считается готовой к выполнению, если все предшественники уже включены в расписание, следовательно, удалены из списка . Другими словами это первая по списку работа, на которой достигается минимум момента готовности по всем готовым работам, т.е.

, (1.15)

где – наименьший момент времени, в который освобождается нужное количество ресурсов необходимое для выполнения работы .

– наименьший момент времени, в который все предшественники работы оказываются завершенными;

Предположим, что равенство нулю или означает отсутствие соответствующего на них ограничения.

Затем найденную работу следует назначить на станок для выполнения в момент времени и удалить из списка . Описанная процедура повторяется до тех пор, пока список не будет пуст.

Во многих случаях удается оценить гарантированную точность такой работы. Большинство полученных оценок определяют верхнюю границу отношения , где – длина расписания, составленного списочным алгоритмом с параметром приоритетов – минимальная длина расписания.

Для нас очень интересны оценки для задач и , они равны соответственно ― и – —. Таким образом, при «неблагоприятных» исходных данным списочный алгоритм получит результат, отличающийся от оптимального почти в два раза. Однако, на практике, как правило, полученные расписания будут близкие к оптимальным [11].

Рассмотрим задачу . Для описания этого процесса воспользуемся спиралевидной циклограммой на рисунке 1.1.

Увы, но вычислительная сложность этой задачи не выяснена, поскольку ни доказательство NP-трудности, ни полиномиальный алгоритм для нее не известны. Однако ее можно решить с помощью приближенного полиномиального алгоритма с гарантированной точностью. Для описания этого алгоритма потребуются описанные ниже, следующие понятия.

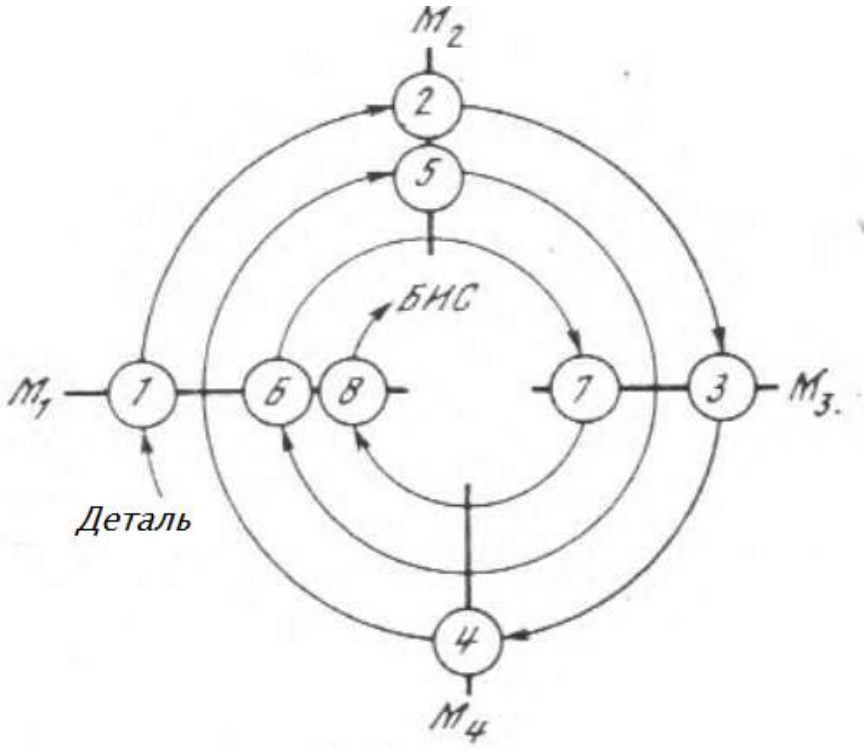

Рисунок.1.1 Пример циклограммы

Обозначим как момент начала операции . Расписание для задачи назовем повторным с периодом , если для всех и выполнено условие

(1.16)

где .

Для составления такого расписания необходимо указать моменты начала операций работы и период , поскольку расписание выполнения остальных работ получается из расписания выполнения работы сдвигами вправо.

1.2.4 Типы алгоритмов

Многие из выборочных, численных и эвристических методов в теории расписаний рассчитывают важность составления огромного количества расписаний для отдельной задачи. Чтобы описать эти методы, существует какое-то количество алгоритмов составления расписаний, а также есть достаточно много различных способов получение компактных расписаний.

Множество состоящее из операций используется во всех алгоритмах составления расписаний, на каждом шаге выбирается одна операция и приписывается момент начала выполнения [5]. Метод начала операции и порядок выбора устанавливают характер алгоритма.

Существует важное различие между корректируемыми и однократными алгоритмами [11]. Для корректируемого алгоритма характерно, что каждый из

уже назначенных моментов начала выполнения можно изменить в ходе дальнейшего составления расписания. Для однократных алгоритмов, момент начала каждой операции остается неизменным в ходе дальнейшего составления расписания. Однако, требуется, чтобы правила выбора очередной операции и момента начала ее выполнения определялась строгой последовательностью в множестве операций и позволяло бы сделать ровно N назначений моментов начала выполнения. Однократными являются почти все алгоритмы, для которых составлены программы на ЭВМ. Корректируемы алгоритмы, тоже можно реализовать на ЭВМ, но формализация правил изменения и регулирования моментов начала операций обычно представляет значительные трудности.

Реализация однократного алгоритма существует для любого расписания, но не нужно ограничиваться этим классом алгоритмов [16]. Оптимальный однократный алгоритм существует, для конкретной задачи, но для другой он может сильно отличаться от оптимального. Для некоторых задач легче использовать корректируемый алгоритм, нежели без предварительных оснований выбирать однократный.

Диспетчеризация является классом однократных алгоритмов и очень важен в теоретических исследованиях и на практике. Характерным для этого класса является то, что последовательно назначаемые моменты начала операций для машин образуют строго возрастающую последовательность. Иначе, решения о включении в расписание принимаются в том порядке в котором будут исполнены и процесс составление можно растянуть во времени, принимая каждое решение перед его исполнением. В классе диспетчеризации каждое расписание можно получить с помощью некоторой диспетчеризации, но можно заметить, что имеет смысл или необходимо рассматривать алгоритмы, не принадлежащие к однократным, так как он является теоретически достаточным.

Не редко при рассмотрении различных алгоритмов важным понятием оказывается множества ожидающих операций. В каждый момент времени множество ожидающих операций, обозначаемое через , есть подмножество тех операций из , для которых предшествующие уже включены в расписание; другими словами, какая-то операция входит в , если уже есть моменты начала выполнения всех предшествующих ей операций данной работы. Это означает, что в начальный момент множество в системе типа состоит из операций – первых в каждой работе. После выбора и включения в расписание одной из этих операций на ее место приходит вторая операция той же работы, если она имеется. Когда последняя операция какой-нибудь работы включена в расписание и ее нечем заменить в , то число элементов множества уменьшается на единицу. Составление расписание можно закончить когда множество опустеет. Множества ожидающих операций важно при исследовании корректируемых и однократных алгоритмов. В корректируемых алгоритмах возможен пересмотр всех сделанных значений и операции могут быть включены по нескольку раз, а в однократных алгоритмах операция покидая не возвращается в это множество.

В основном используют два способа разбиения . Первый способ заключается в том, что в возникает *m* подмножеств, соответствующих *k* машинам и обозначаются S_{so}^k , . Все подмножества состоят из операций, выполнение которых осуществляется определенной машиной и которые можно включить в расписание в данное время. Другими словами, подмножество S_{so}^k – это операции ожидающие выполнения на машине *k* и таких, что предыдущие операции уже будут включены в расписание и выполняются другими машинами. Изначально имеем *n* операций разбитых на *m* подмножеств S_{so}^k , . В процессе составления расписания подмножества могут становиться то непустыми, то пустыми и пополняться до тех пор, пока все подмножества не станут пустыми и составление расписания будет окончено.

Большой интерес представляют два метода включения операций в подмножество S_{so}^k , из которого выбирается операция для включения в расписание. Каждый из этих методов можно привести к определенному классу диспетчеризаций. Возьмем множество «выполняемых» операций и обозначим его . Операция как только покидает сразу попадает в множество . Оно разбивается на *m* подмножеств, обозначаемых S_{so}^k , , пропорционально количеству машин, и в каждый момент времени существует по одной операции в каждом из подмножеств (операции, выполняемые в этот момент каждой машиной). Предыдущая операция покидает подмножество при включении очередной операции в одно из множеств S_{ip}^k .

Второй способ – это разбиения на n подмножеств, соответствующих работам; тогда все подмножества состоят из минимум одной операции и становятся пустым как только последняя операция работы включается в составленное расписание. В алгоритме поочередного включения работ используется такой способ разбиения, при котором все моментов начала операций одной работы производятся последовательно, не чередуются с моментами начала операций других работ. Поскольку существует перестановок работ, если ограничится только алгоритмами, дающими компактные расписания, получим ровно однократных алгоритмов такого типа и каждый из них приводит к единственному компактному расписанию, но эти расписания не обязательно различный, например если некоторые работы просят разных комплектов машин.

Иногда моделированием процесса называется составление расписаний на ЭВМ с использованием одного из алгоритмов диспетчеризации. Это объясняется тем, что программа воспроизводит физический процесс выполнения операций во времени в том смысле, что назначенный порядок совпадает с порядком реального выполнения операций.

Понятие приоритета операции или работы очень важно для нас [17]. Приоритет – это числовая характеристика операции или работы. Допустим в алгоритме последовательного включения работ необходимо правило, которое определит порядок включения работ в расписание, и это правило формулируется в терминах приоритетов: работы выбираем в порядке возрастания их числовых характеристик. Всем из n! способов назначенные приоритеты работ будет соответствовать один определенный алгоритм указанного типа – алгоритм последовательного включения работ.

Система приоритетов должна быть достаточно полной, для того чтобы две «конкурирующие» операции (работы) имели различные приоритеты и был бы возможен однозначный выбор. Иначе нужно будет ввести вторичные приоритеты на случай равенства основных. Допустим, в алгоритме поочередного включения работ число входящих операций можно выбрать в качестве первичного приоритета. Так как могут совпадать числа операций некоторых работ, необходимо задать вторичные приоритеты и из таких работ в первую очередь выбрать то, что имеет минимальный вторичный приоритет.

С помощью некоторого вероятностного правила можно получить общий метод задания приоритетов с использованием различных распределений для различных наборов работ и операций [18]. Используя такой подход особо внимание уделяем идеи множественного составления расписания для одной задачи. Связано это со случайностями при вычислении приоритетов и каждый будет приводить к разным расписаниям. Смысл составление повторного расписания в том, что, во-первых, что, выбирая множество работ, получаем статические выводы для расписаний определенного класса систем (1 вариант), а во-вторых, используя стохастическое правильно назначения приоритетов и получая множество расписаний той же задачи, делаем статические выводы для правил назначения приоритетов и расписаний для этой системы (2 вариант).

Назначая приоритеты используют невырожденное или вырожденное распределение, но только одно и тоже для всех работ. В вырожденном распределении значение приоритетов фактически постоянные, и они могут совпасть с полной длительностью ее выполнения. Тогда повторное составление расписания с 2 вариантом бессмысленно, потому что будут получаться одинаковые расписания. Используя невырожденные распределения не имеют большого смысла, так как приводят к равновероятному выбору операций (работ) из множества всех возможных. В этом случае оба варианта можно решить многократно.

Любая диспетчеризация предусматривает необходимость какого-нибудь приоритетного правила, чтобы назначить очередную операцию, выполняемую машиной. Не всегда приоритеты задаются неявно в вычислительной программе, но они всегда существуют. Два этапа происходит выбор операции. Сначала выбираем машину k, а затем из множества S_{so}^{k}, выбираем операцию для включения в расписание. Затем, когда выбрана машина, можно выбирать

операцию в порядке поступления. В общем случае приоритеты задаются с помощью стохастического правила, используя разные распределения, в частности, сосредоточится в одной точке (допустим, выбор в порядке поступления), или выбирать любую другую из возможных операций. Если распределение вырожденное, то получим единственное расписание, и повторно составлять по 2 варианту бессмысленно. Следовательно, в общем случае диспетчеризация является алгоритмом с вероятным правильном назначения приоритетов; лишь иногда приоритеты и вероятности не включаются в алгоритм явно. Поэтому диспетчеризация устанавливает определенный алгоритма составления расписания, основанный на стохастических правилах назначения приоритетов.

Не много другой подход к рассматриваемым вопросам рассмотрен Шрейджем, Джеремиа, Лалчалдани. Выбор операции из хаотичного множества производится на основании характеристик работ или операций. Тогда момент начала выполнения выбранной операции происходит так, чтобы получить сжатое расписание. Рассмотрим правила выбора операции из :

– SPT – выбор операции с наименьшей длительностью;

– LWKR – выбор операции у которой длительность выполнения всех операций минимальна;

– MWKR-P – выбор операции у которой длительность выполнения всех остальных хаотичный операций максимальна;

– MWKR/P – выбор операции у которой максимальное отношение длительности всех хаотичных операций к длительности операции, включенной в расписание;

– RANDROM – случайный выбор;

– MWKR – выбор операции у которой оставшаяся длительность всех операций максимальна;

– MOPNR – выбор операции с максимальным числом хаотичных операций.

Полученные по этим правилам расписания сравнивали с не задерживающими расписаниями, полученные с помощью диспетчеризацией: сначала выбирали машину, которая начнет выполнение очередной операции, а затем выбирали операцию по одному из правил:

– LPT – выбор наиболее длинной операции;

– FCFS – выбор операции в порядке поступления;

– MWKR – выбор операции наибольшей длительностью выполнения всех хаотичных операций;

– LWKR – выбор операции с наименьшей длительностью выполнения всех хаотичных операций;

– RANDROM – случайный выбор;

– SPT – выбор операции с наименьшей длительностью выполнения.

Можно увидеть, что правила из этих двух списков одинаковы, различие только между выбором из подмножества операций и из всего множества , которые ожидают выполнения на машине, которая освободится раньше других, это приводит к абсолютно разным расписаниям.

1.2.5 NP-полнота

Большая часть задач, имеют полиномиальные алгоритмы решения [13]. Это значит, что время работы алгоритма на входе длины *n* составляет не более $O(n^k)$ для какой-то константы *k* (не зависит от длины входа). Конечно, не каждая задача имеет алгоритм решения, выполняющее это свойство. Некоторые задачи вообще могут не решаться. Например, задача «проблема остановки» выясняет останавливается программа на входе. Еще бывают задачи, у которых существует решающий их алгоритм, но работает он очень долго.

Если мы хотим провести формальную границу между практическими алгоритмами и алгоритмами, представляющими лишь теоретический интерес, то класс алгоритмов, работающих за полиномиальное время, является разумным приближением. Мы рассмотрим класс задач называемый NP-полными [6]. Для них не найдены полиномиальные алгоритмы, но не доказано, что их не существует, Изучение NP-полных задач связано с так называемым вопросом «P = NP». Это является сейчас одной из наиболее сложных проблем теории вычислений и был поставлен 1971 году.

Чтобы изучать понятия разрешимая «разрешимая проблема», «алгоритм», нужно ввести формальное устройство, по которому можно доказывать математические факты. Исторически для этих целей была выбрана машина Тьюринга, описанная Аланом Тьюрингом в 1936 году [7].

Информация, подаваемая на вход машине Тьюринга, хранится на ленте, бесконечной в обе стороны. Во всех клетках ленты хранится один символ из специального алфавита входных символов. Входные данные располагаются на ленте одной группой, а все остальное пространство заполнено зарезервированными символами – пробелами.

В один промежуток времени одна из клеток выделяется потому что на эту клетку указывает ленточная головка. В начальный момент времени головка указывает на первый (не пробельный) символ входных данных.

Также в каждый момент времени машина Тьюринга в одном из конечного множества состояний. Одно из таких множеств является начальным, а некоторые множества состояний помечены как допускающие. Именно так машина Тьюринга сообщит по завершению свой работы о полученном результате.

Каждый дискретный шаг происходит следующим образом. Машина Тьюринга имеет в данный момент некоторое состояние, а ленточная головка указывает на некоторый символ и для этой пары однозначно определена соответствующая ей тройка: новый символ, новое состояние, сдвиг. Следовательно, машина Тьюринга переходит в новое состояние, а на ленте вместо прочитанного символа оставляет

новый символ, и ленточная головка сдвигается вдоль ленты либо вправо на одну позицию.

Недетерминированная машина Тьюринга (НМТ) отличается от обычной следующими правилами:

– НМТ выбирает любой переход;

– для каждой пары (состояние, символ) определены несколько переходов;

– вход допускается НМТ, если хотя бы одна последовательность переходов приводит НМТ к допускающему состоянию.

Проблема принадлежит классу NP, если существует НМТ, которая разрешает ее за полиномиальное время.

Вывод

Рассмотренные типы алгоритмов и задач позволяют сделать вывод о применимости методов к решению поставленной задачи. Ниже приведен подробный анализ. В данной главе подытожим. Все предлагаемые в литературе модели имеют упрощенный характер, а все эффективные алгоритмы составлены для конкретных задач, которые не учитывают особенности реальных производств. Исходя из этого, целесообразно разработать эффективный алгоритм, который будет составлять производственное расписание, опираясь на известные алгоритмы диспетчеризации. Важно учесть все ограничения, накладываемые на решение. Решение следует искать в комбинированном подходе, использующем несколько эвристик, но они должны учитывать особенности технологических процессов на предприятии. В конечном итоге поиск решения ограничим областью алгоритмов диспетчеризации, а использование эвристик в алгоритме говорит нам о модифицировании классической модели составления расписания.

2 МАТЕМАТИЧЕСКАЯ МОДЕЛЬ И АЛГОРИТМ СОСТАВЛЕНИЯ РАСПИСАНИЯ

2.1 Предварительная постановка задачи

В соответствии с поставленной задачей нужно, опираясь на теоретические знания по теории расписаний, произвести ряд математических процедур, которые смогут решить задачу составления производственного расписания выпуска продукции. При это нужно понимать, что разрабатываемая система должна удовлетворять следующим требованиям.

Решение, которое будет разработано должно удовлетворять требованиям быстродействия, в зависимости от конкретного типа задачи решение будет получено за определенное время. Еще предполагается, что объем данных только будет расти, поэтому система должна удовлетворять требованиям «с запасом».

На сведения о конкретных производственных процессах на предприятии опираются разрабатываемые алгоритмы. Алгоритмы существенно зависят от конкретной задачи.

На основе разработанной модели составить программу, которая решала бы три задачи:

– учет отказа оборудования;

– задачу составления расписания, которая представляется в графическом виде;

– для исходной информации использовать данные из среды Visual studio C#, вывод данных производить средствами данной среды разработки.

2.2 Выбор методов и предварительный анализ задачи

Требования, которые мы предъявляем к системе составления расписания, могут определить те классы методов, из которых следует брать для разработки алгоритмов.

Мы можем заметить, что решение конкретной задачи очень сильно зависит от набора данных, так что необходимо в рамках алгоритма классифицировать задачу, что иногда не представляется возможным.

Еще очень важно при разработке алгоритма, какой объем данных у нас имеется для решаемой задачи, поэтому исходя из объема для решения используются различные подходы.

Естественно в условиях ограниченности времени выполнения программы и при определенных объемах данных следует использовать ряд эвристических подходов.

Следует выбирать известные алгоритмы для классифицируемых задач в качестве базовых алгоритмов, а также нужно анализировать исходные данные и принимать решение о применимости конкретного метода. Чтобы решить не

классифицируемые задачи необходимо использовать комбинированный подход, сначала расписание составляется по предположению об эффективности подхода к решению задачи, а затем поиск псевдооптимального решения с использованием эвристического подхода.

Полученное расписание должно удовлетворять требованиям выполнимости, с учетом возможных трудностей возникшим на реальном производстве, это требование дает нам неоптимальное расписание.

Поставленная задача не сводит нас к составлению статического расписания. Известно, что технологический процесс предполагает заминки на производстве и в силу таких причин накладывается дополнительные ограничения на быстродействие в случае оперативного пересчета расписания.

2.3 Моделирование системы составления расписания

2.3.1 Математическая модель системы составления расписаний

Понятие операции является основным понятием теории расписаний. Операция – это элементарная задача, подлежащая выполнению. Операцию можно охарактеризовать:

- индексом принадлежности к определенной работе;
- числом, представляющим собой объем операции;
- индексом принадлежности к определенному прибору.

Работами называется система непересекающихся подмножеств, которая разбита на множество операций по первому индексу. Разбиение исходного множества по второму индексу приводит к непересекающимся подмножествам операции. Формально опишем следующим образом:

Множество работ (требований), необходимо выполнить без остановки на нескольких или одном приборах M_i, где , они выполняют в каждый момент времени не более одной операции. Время выполнения операции на приборе M_i, обозначают . Чтобы требование k стало доступным для выполнения, необходимо задать время поступления r_k. Операция k может иметь директивный срок выполнения d_k – эта величина показывает момент, у которого операция должна выполниться, а также вес w_k, введем v_k – критерий, который определяет включение операции в расписание.

Отношения предшествования можно задать между операциями в виде ацикличного ориентированного графа G. У каждой работы можно задать последовательность составляющих ее операций. Если операция X должна быть осуществима раньше, чем Y, то тогда X предшествует Y. Просто запишем это в виде: $X<Y$. Отношение порядка транзитивно, другими словами если $Y<Z$ и $X<Y$, тогда $X<Z$, и дуги изображают отношение непосредственного предшествования, а вершины графа операции. Если существуют пути между вершинами, то они связаны отношением порядка.

Устройство, которое может выполнить все, что связано операцией будем называть машиной. Множество всех машин, используемых для выполнения множества операций – система обслуживания.

Время окончания выполнения требования *k* при расписании π обозначим через $C_k(\pi)$. Обозначим L_k как временное смещение требования k в расписании π

, (2.1)

также определим

(2.2)

как его запаздывание. обозначает запаздывает ли требование *k* в расписании π или нет: , если k запаздывает, иначе значит *k* выполняется вовремя. Для составленного расписания в качестве критериев обозначим:

, (2.3)

где π_j *П(N).*, *П(N).* – множество допустимых расписаний, ; уменьшение максимального времени окончания выполнения (задача на быстродействие),

, (2.4)

уменьшение числа запаздывающих операций.

Через π_i обозначим расписание из множества элементов *N*, выполняемых на приборе M_i, где , множество требований в расписании обозначим

, (2.5)

а через

, (2.6)

где ;

расписание для всего множества требований . Будем рассматривать только возможные расписания без искусственных простоев оборудования, подходящим отношениям предшествования. Множество допустимых расписаний не пусто для любого ацикличного графа *G*.

Момент окончания выполнения требования *k* на приборе при расписании определяется как

(2.7)

где – множество требований предшествующих требованию k, согласно графу G;

– требования, согласно расписанию на приборе , предшествующие обслуживанию требования .

Множества требований N выполняют множество расписаний, допустимых относительно графа предшествования *G*, будем обозначать через .

Чтобы составить расписание нужно, чтобы для каждой операции k на временной оси задается отрезок, когда это операцию нужно выполнить машиной M_i, получается, что для каждой операции один или более интервалов таких, что:
при этом должно выполняться следующее условие, что если $X<Y$, то $b_{LX}<a_{1Y}$ b и каждый из интервалов (a_j,b_j), находится целиком внутри одного из отрезков времени доступности соответствующей машины. В итоге составленное расписание может рассматриваться как задача упорядочения операций, выполняемых каждой машиной.

Рассмотрим следующие существенные ограничения.

1. Работы состоят из строго упорядоченной последовательности операций.
2. Каждая операция выполняется только одной машиной.
3. Каждая машина может быть назначена в любой момент времени, это значит, что запрещены перерывы в их работе. И каждая машина формально представляет интервал (0, T).
4. В каждый момент времени машина может выполнять только одну операцию.
5. Для каждой операции задан один интервал (a, b), причем их длительность равна (b – a).
6. Интервалы выполнения последовательных операций одинаковых работ не пересекаются.
7. Допустим, что p_{ik} включает в себя полные настройки машины, предыдущей выполненной операции, и все переналадки ее после выполнения операции. Такое предположение равносильно тому, что длительности настройки машины не зависят от последовательности операций, это значит, что время необходимое для подготовки машины к выполнению некоторой операции, не зависит от того, какую операцию последней выполнила машина.

2.3.2 Выполнение ограничений

Исходя, что каждая машина в один момент времени выполняет одну работу, следует, что для каждой пары работ (операций одной работы) k_i и k_j выполняется только одно из неравенств:

, , , (2.8)

где начальное время выполнение операции l работы k на приборе M_i;

начальное время выполнения операции j работы k на приборе M_i;

выполнение работы k_l идет после выполнения работы k_j или

, , , (2.9)

выполнение работы k_j идет после выполнения работы k_l.

Нельзя формально описать ограничение типа «или-или» и требуется ввод целочисленных переменных:

После того как мы сформулировали ограничение типа «или-или», запишем его в виде двух условий, каждое должно быть выполнено:

, (2.10)

, (2.11)

где большая константа, выбранная так, чтобы могло выполнится только одно из двух условий: или .

Пусть k_l идет после работы k_j, то есть и , тогда (2.11) совпадает с (2.9), а (2.10) благодаря параметру M превращается в избыточное ограничение, которое не противоречит системе в целом.

Опишем формально ограничения на порядок выполнения операций. Заметим, что

, (2.12)

где ;

момент начала выполнения операции работы .

Тогда для всех операций каждой работы, должно иметь место неравенство

. (2.13)

Графически это представлено на рисунке 2.1 следующим образом:

Рисунок 2.1 – Последовательность выполнения операций

2.4 Выбор методов и анализ исходных данных

2.4.1 Анализ исходных данных

В этом разделе рассмотрим конкретные исходные данные, и ссылаясь на эти данные сделаем вывод о применение различных подходов к решению поставленной задачи. Классификация задачи в существующей модели дает использовать известные методы решения для специфичных данных. Если задачи не имеют известных решений, решение их будем получать, опираясь на эвристические методы.

Для описания конкретных постановок задач используется трехпозиционная формула , предложенная в работе [2]. В этой формуле определим параметры , , , зная представления о конкретных значениях параметров.

Для еще одной классификации задач проведем временные ограничения на работу системы составления расписания, это даст нам применить метод «полного перебора» для получения оптимальных расписаний, когда временные ограничения позволяют это сделать.

Чтобы определить параметр , определим сначала и , при этом . Первая позиция определяет особенности использования приборов.

В частности, из (1) следует, что необходимо определить выполняемые работы на каждом станке, а также необходимо определить является система станков последовательной и параллельной. Ниже, на рисунке 2.2, показана часть технологического цикла одной из работ:

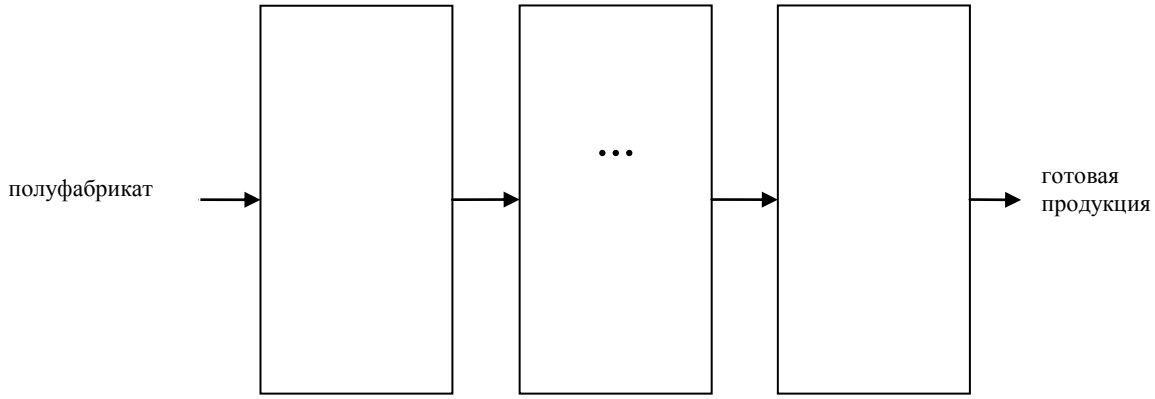

Рисунок 2.2 – Технологический процесс

Из рисунка можно увидеть, что последовательная система станков выполняет операции , представляющие собой цепи операций , в которых операция выполняется на приборе с номером в течении времени . Запись означает, что операция не может выполняться до завершения операции , причем для всех *j* . Делаем вывод что, детали могут проходить обработку по разным назначенным маршрутам, причем любые два соседних станка в одном маршруте обязательно различны. Из вышесказанного .

Значение параметра обозначает количество в системе станков, то есть .

Определим параметр , для этого, согласно (2) необходимо определить параметры , , .

Параметр определяет особенности выполнения работ. , а именно ресурсные ограничения. Для упрощения модели мы их не учитываем, поэтому значит ресурсные ограничения отсутствуют.

Параметр определяет отношение порядка, заданное на множестве работ, так как порядок важен для операций конкретной работы, то в качестве значения параметра выберем значит на множестве работ задан порядок .

Параметр определяет длительность операций работ, из рисунка 2.2, можно увидеть, что длительности операций зависят только от производительности станков значит .

Третья позиция показывает критерий эффективности расписания. Для критериев выберем следующие: C_{max} длина расписания или максимальный момент завершения. Максимальным по быстродействию называют расписание минимизирующее C_{max}. Взвешенное число опозданий – .

В конечном итоге получим задачи , .

Очевидно, что для небольших объемов исходных данных, а именно количестве выполняемых работ допустимо использование алгоритма полного перебора, с учетом ограничений, накладываемых на расписание.

Еще одной задачей в рамках системы составления расписания является оценка вновь поступающих требований, помогающая определить директивный срок выполнения работы d_k.

2.4.2 Выбор методов решения

Мы определили 4 типа основных задач, возникающих при решение поставленной задачи.

Для начала рассмотрим задачу . В настоящее время эффективных алгоритмов не разработали и для значений m задача NP – полная [15]. Для случая двухоперационных работ попробуем адаптировать алгоритм конвейерной задачи, то есть в рамках рассмотренной модели получим N_k работ, представляющие цепи операций , в которых операции A_k и B_k выполняются на приборе с номером в течении времени . Пример приведен на рисунке 2.3:

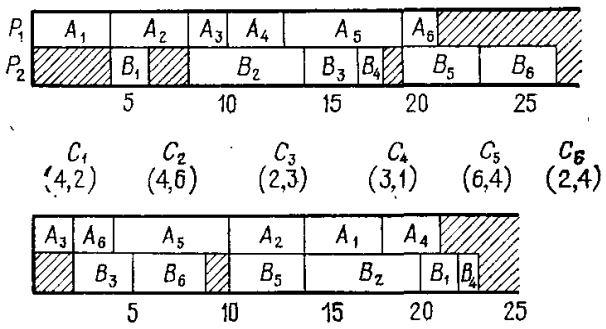

Рисунок 2.3 – Два конвейерных расписания

На рисунке 2.3 задания выполняются на в том же порядке, в котором выполняются на . Покажем, что ограничимся рассмотрением расписаний, в которых если выполняется перед , то и также выполняется перед .

Допустим S – расписание на двух процессорах для набора заданий , где каждое задание состоит из операций и , скажем, что задание выполняется раньше задания , тогда и только тогда, когда выполняется раньше, чем и – раньше, чем . Согласно лемме из [14], с учетом пары допущений получим утверждение, что можно построить расписание не длиннее, чем S, в котором для каждой пары и либо выполняется раньше , либо выполняется раньше .

Исходя из выше рассмотренного, можно ограничиться расписаниями, в которых задания на приборе выполняются в том же порядке, что и задания на приборе . Для того, чтобы решить задачу, рассмотрим две работы и . Если выполняется перед , то длина промежутка времени

(2.14)

где

, , , – время выполнения операций работ и соответственно.

Аналогично (2.15) получим

По лемме из [19] получим, что условие, при котором короче , является

(2.15)

Это утверждение позволяет на основе следующей леммы из того же источника [19] утверждать, что можно построить расписание минимальной длины в котором, если выполняется (2.16), то работа выполняется перед . Делаем вывод, что с помощью условия (2.16) можно частично упорядочить работы. Осталось зафиксировать правила разрешения конфликтов и получим полностью упорядоченную последовательность.

Теперь рассмотрим то же утверждения для случаю трехоперационных работ, у которой. в рамках рассмотренной модели получим работ, представляющие собой цепи операций , в которых операции , и выполняются на приборе с номером в течении времени . Если

или если , где

– время выполнения операций работ и соответственно, то мы можем использовать условие

(2.16)

чтобы поместить работу перед работой .

2.5 Модификация алгоритма

В качестве упорядоченного множества возьмем множество, которое состоит из операций. Основная идея алгоритма заключается в следующем: на шаге выбирается одна операция и ей дается момент начала выполнения. Характер нашего алгоритма определяет порядок выбора и момент начала операции.

Для основы возьмем однократный алгоритм, он описывается тем, что в выбранный момент начала каждой операции остается неизменным в ходе дальнейшего составления расписания. При этом требуется, чтобы правила выбора очередной операции и назначения момента начала ее выполнения определялись бы строгой последовательностью в множестве операций и это позволило бы нам сделать ровно N назначений моментов начала выполнения.

Для любого расписания существует реализующий его однократный алгоритм [16]. Для конкретной задачи, существует оптимальный однократный алгоритм, однако для другой он может сильно отличаться от оптимального. Этот момент рассмотрим позднее. Реализуем класс однократных алгоритмов, который называется диспетчеризацией. Характерным для этого класса является, последовательные назначаемые моменты начала операций для каждой машины образуют возрастающую последовательность. По-другому, решение о включении в расписание принимаются в том же порядке, в котором их приведут в исполнение. Класс однократных алгоритмов достаточен, как и класс диспетчеризаций: каждое расписание можно получить с помощью некоторой диспетчеризации.

Понятие множества ожидающих операций оказывается важным. В каждый момент времени множество ожидающих операций, обозначается через , есть подмножество тех операций из , для которых следующие уже включены в расписание; другими словами, некоторая операция входит в , если уже выбраны моменты начала выполнения всех следующих ей операций данной работы. Это означает, что в начальный момент множество в системе типа состоит из операций – первых в каждой работе. После выбора и включения в расписание одной из этих операций ее место занимает вторая операция той же работы если она есть. Каждый раз, когда последняя операция какой-либо работы

включена в расписание и ее нечем заменить в , число элементов этого множества уменьшается на единицу. Составление расписания является законченным, когда множество становится пустым.

Понятие приоритета операции или работы очень важно. Приоритет – это числовая характеристика операции или работы. Допустим в алгоритме последовательного включения работ необходимо правило, которое определит порядок включения работ в расписание, и это правило формулируется в терминах приоритетов: работы выбираем в порядке возрастания их числовых характеристик. Всем из n! способов назначенные приоритеты работ будет соответствовать один определенный алгоритм указанного типа – алгоритм последовательного включения работ.

Система приоритетов должна быть достаточно полной, для того чтобы две «конкурирующие» операции (работы) имели различные приоритеты и был бы возможен однозначный выбор. Иначе нужно будет ввести вторичные приоритеты на случай равенства основных. В качестве одного из приоритетов работы выберем число входящих в нее операций. И выбирается машина, которая может начать выполнение операции раньше других, а затем из множества ожидающих операций с максимальным приоритетом выберем операции, согласно правилу MWKR – выбор операции, с максимальной длительностью выполнения всех последующих операций.

Если несколько таких операций, то выбор производится случайным образом и ставится специальная пометка. Для такого подхода есть смысл повторного составления расписания. Для того чтобы, случайности при вычислении приоритетов каждый раз не приводили к другому расписанию. Еще фактор повторного составления расписания, что из них можно выбрать наилучшее расписание.

Выбор операции происходит в два этапа. Сначала выбирается машина, скажем , а затем из множества S_{so}^{k} , состоящего из нескольких операций, выбирается операция для очередного включения в расписание.

Для множества на каждой итерации приоритет назначается согласно нижеперечисленным следующим правилам.

Исходя из количества операций в работе назначаем приоритеты.

Для двухоперационных работ и , , :

, где , , , – время выполнения операций работ и соответственно. Если это условие выполняется, то мы можем поместить работу перед работой .

Для трехоперационных работ и , , .

Если или если , то

рассмотрим условие ,

где , , , , – время выполнения операций работ и соответственно. Если это условие выполняется, то мы можем поместить работу перед работой .

Назначим приоритет также согласно принципу MWKR – зависимость приоритета, от длительности выполнения всех последующих операций работы, которой она принадлежит. Это означает, что больший приоритет операции с большей длительностью последующих за ней операций.

Схема описанного алгоритма имеет вид, представленный ниже на рисунке 2.4.

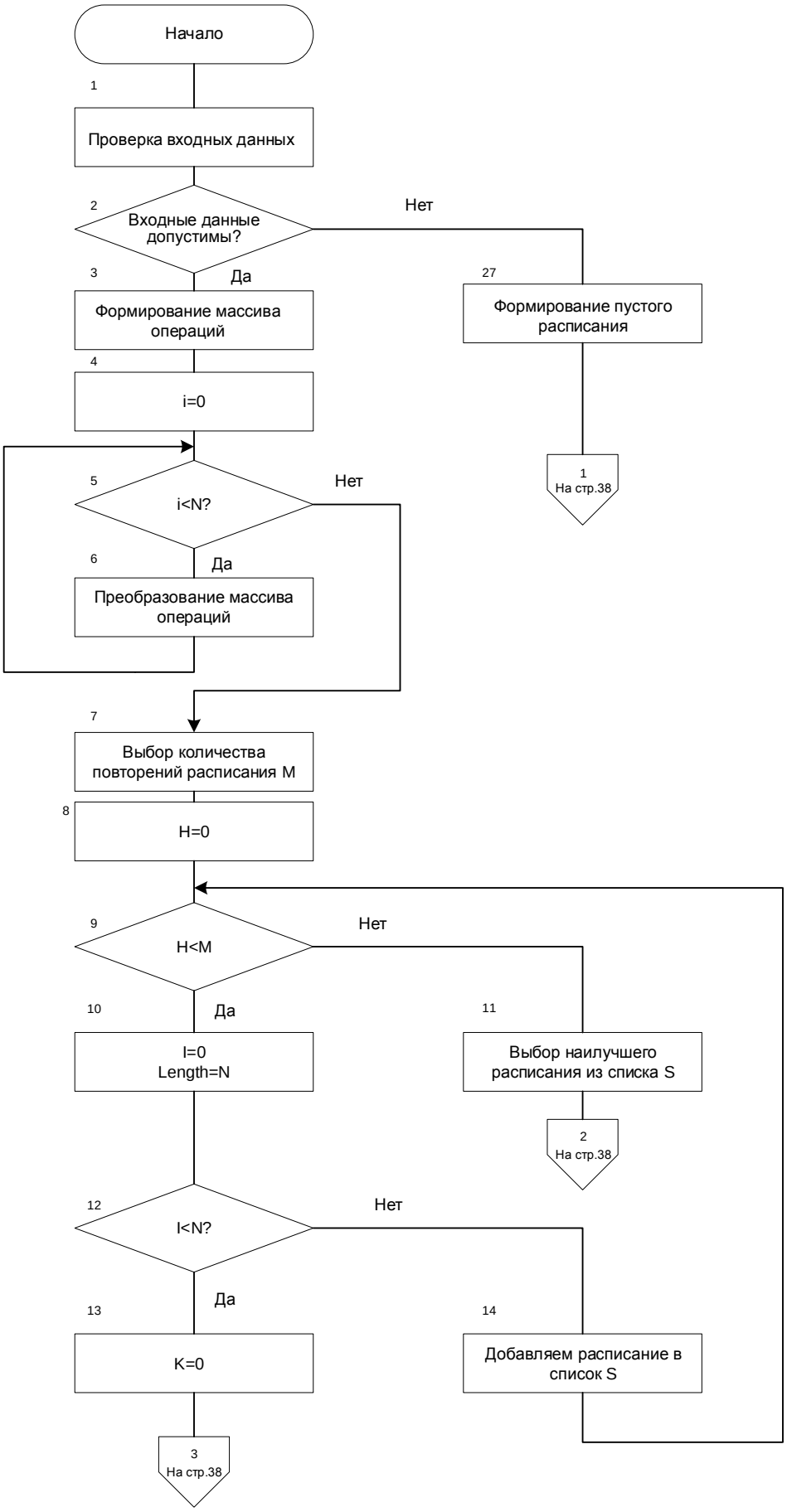

Рисунок 2.4 – Схема алгоритма составления расписания(начало)

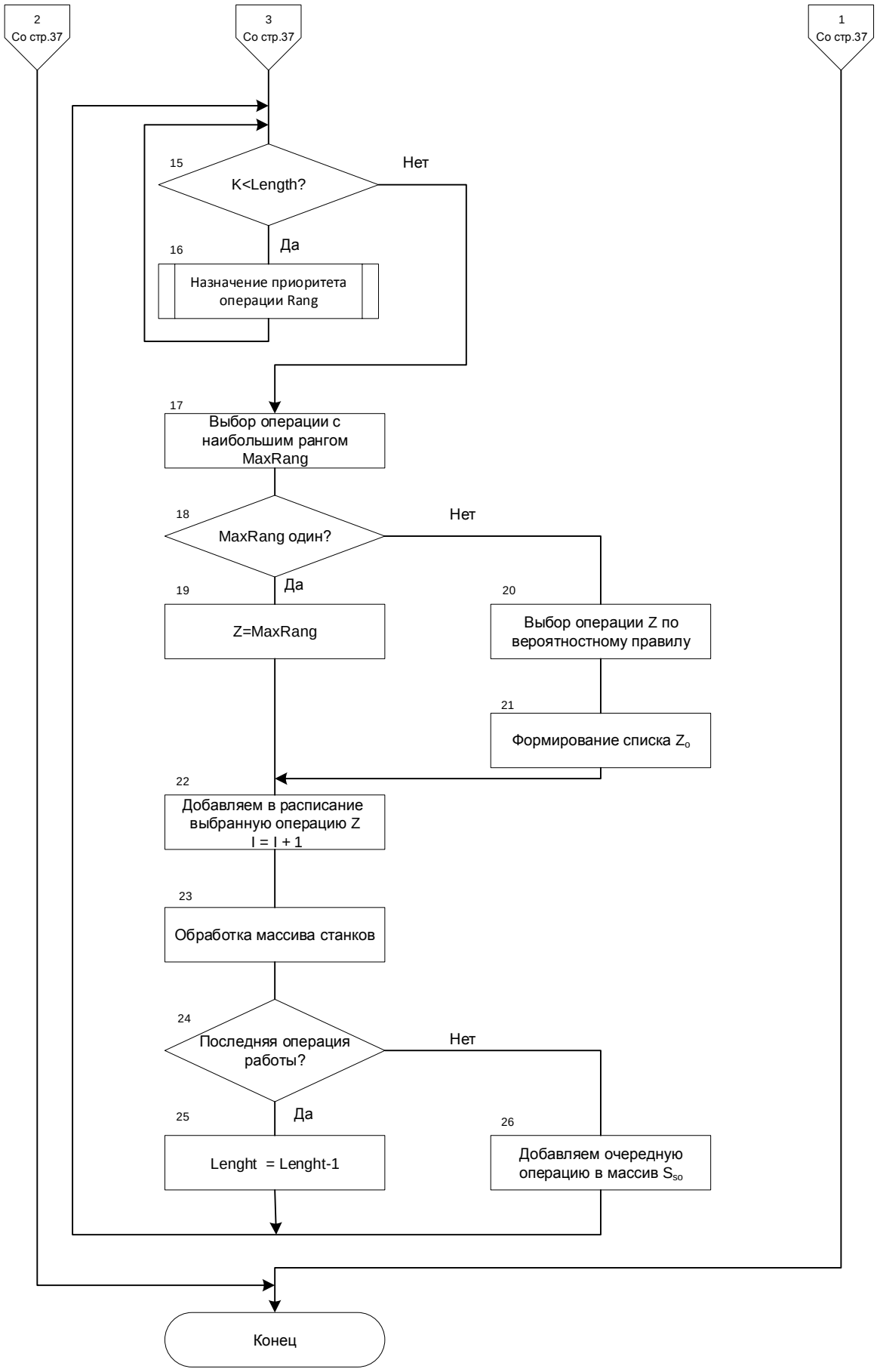

Рисунок 2.4 – Схема алгоритма составления расписания (продолжение)

Вывод

В соответствии с поставленной задачей, опираясь на теоретические знания по теории расписаний, производя ряд математических процедур, выбираем задачу производим анализ данных, составляем математическую модель и выполняем ограничения, которые возникают у математической модели. На сведения о конкретных производственных процессах на предприятии опираются разрабатываемые алгоритмы. Анализируем существующие алгоритмы и разрабатываем свой алгоритм составления расписания, а также описываем его работу. Разработанный алгоритм удовлетворит требованиям быстродействия, в выбранной задаче, решение будет получено за определенное время. Предполагается, что объем данных будет увеличиваться, поэтому система удовлетворит требованиям «с запасом».

3 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

3.1 Оборудование и программное обеспечение

При проведении всех экспериментов по оценке качества работы алгоритмов использовался ПК со следующими характеристиками и ПО:

- Процессор AMD Ryzen 5 1600
- 16 GB RAM
- ОС Microsoft Windows 10, 64-х битная

3.2 Описание программного обеспечения

Приложение разрабатывалось для операционной системы Windows. Разработка программы осуществлялась в среде Microsoft Visual Studio 2017, на языке C#.

Программа состоит из пяти файлов: Algorithm.cs, Colors.cs, InputBoxProvider.cs, MainForm.cs, Schedule.cs.

В Algorithm.cs происходит работа основного алгоритма программы, а также количество повторений расписания и учет отказа оборудования.

В Colors.cs происходит выбор цвета диаграммы.

InputBoxProvider.cs файл в котором задается размеры таблицы, а также создается сама таблица, где заполняются данные расписания.

Schedule.cs файл в котором задается размер диаграммы, а также блоков для расписания.

MainForm.cs основной файл в котором происходит заполнение данных и вывод полученной диаграммы.

3.3 Разработка пользовательского интерфейса

После запуска приложения открывается главное окно программы (рисунок 3.1)

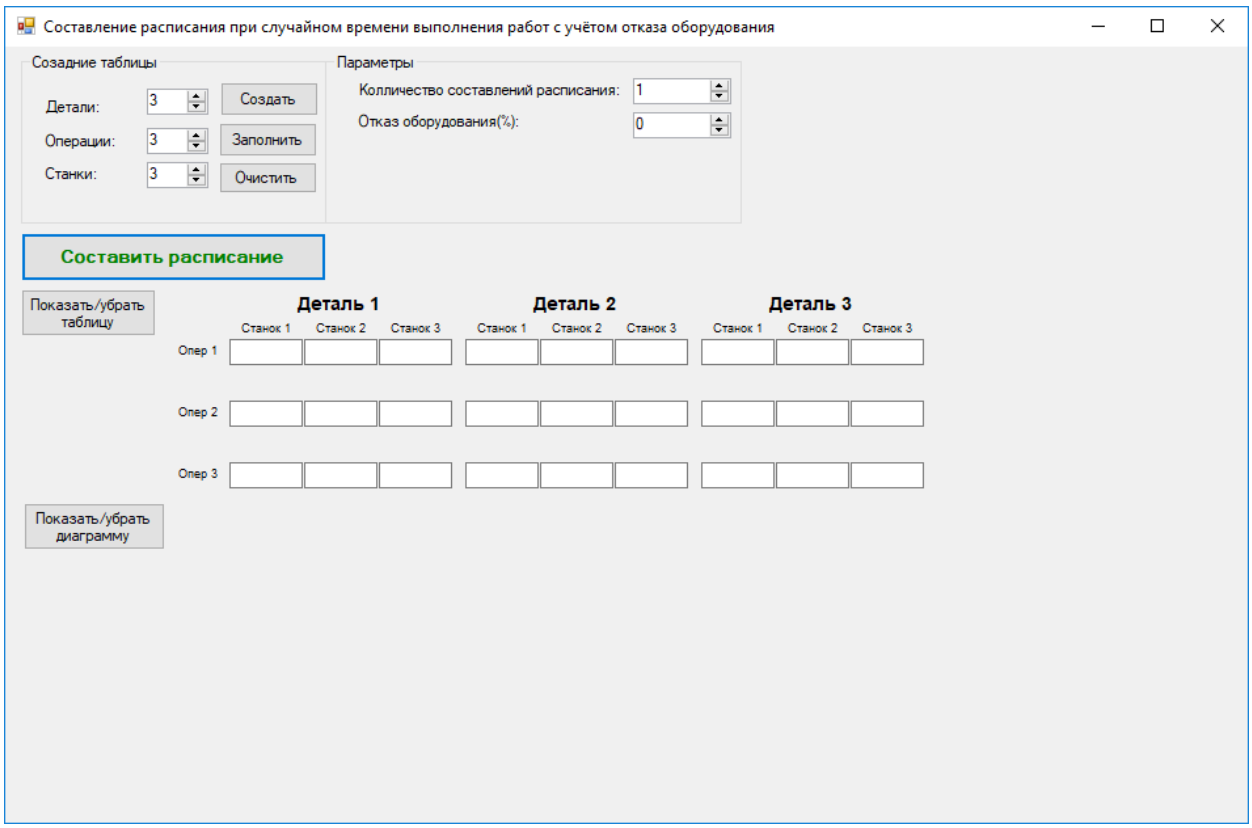

Рисунок 3.1 – Интерфейс главного окна программы

Изменяя количество деталей, операций и станков и нажав на кнопку «Создать», происходит создание таблицы размеров которых указали ранее (рисунок 3.2).

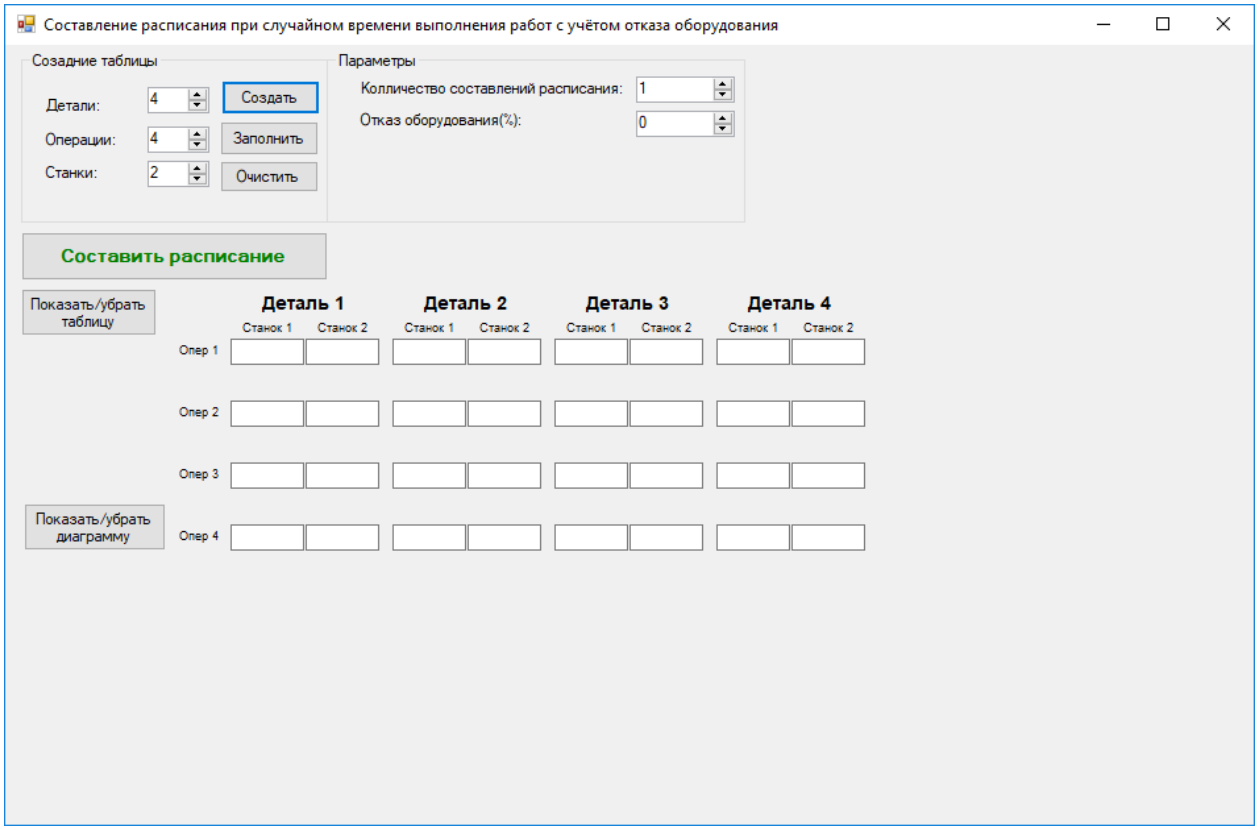

Рисунок 3.2 – Создание таблицы данных для расписания

Данные в таблицу для составления расписания можно заполнить самостоятельно или нажать кнопку «Заполнить», в ячейки вводятся цифры и знак X, при попытке ввести буквы выдаст ошибку (рисунок 3.3)

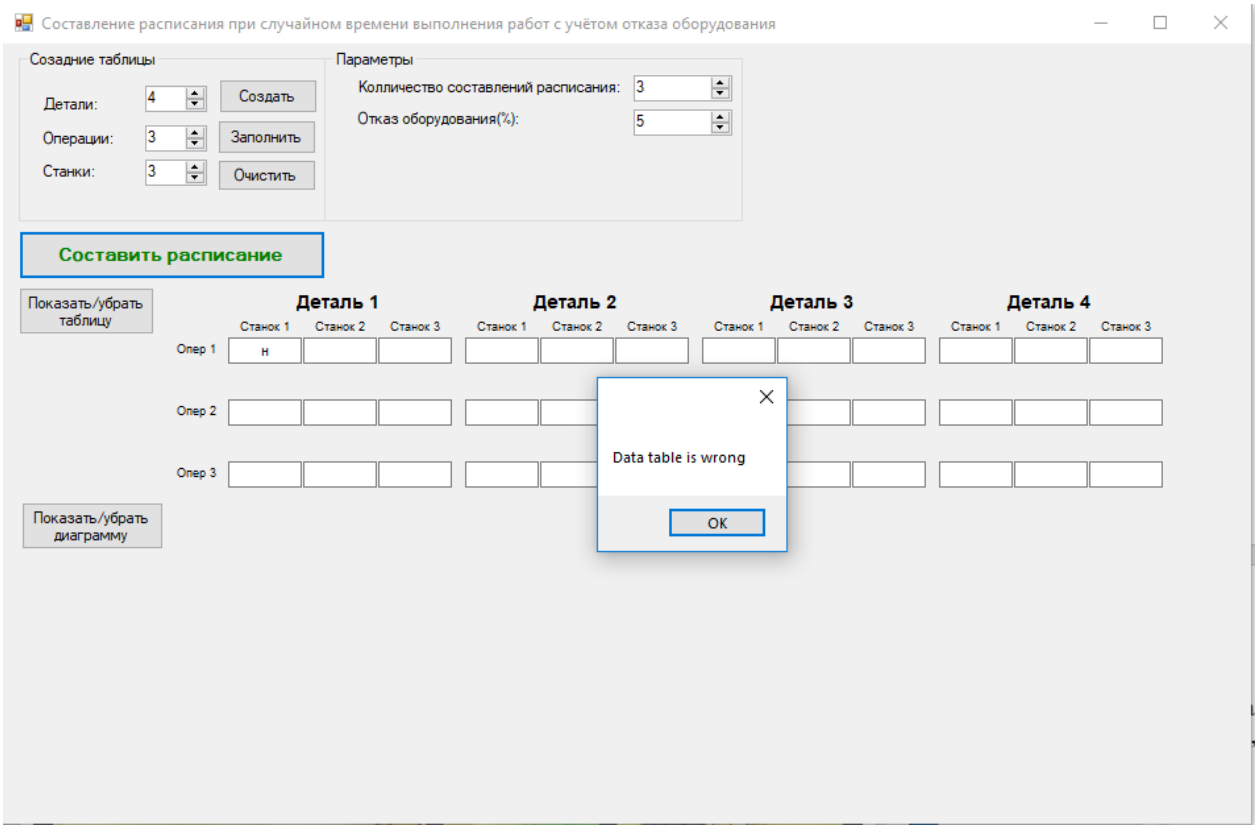

Рисунок 3.3 – Окно ошибки при вводе не правильных данных

На главном окне программы есть два параметра, количество составлений расписания и отказ оборудования. После создания таблицы, заполнения ее и выбора двух параметров, нужно нажать кнопку «Составить расписание», а затем кнопку «Показать/убрать диаграмму».

3.4 Программный эксперимент

После разработки и реализации алгоритма и интерфейса над полученной программой производилось тестирование, для подтверждения ее работоспособности.

Тестирование программы проводилось для 3-х операций и 4-х операций.

Количество станков, которым поступают детали, берем равное 3, количество поступающих деталей равно 4. Количество повторений составления расписания равно 3. Вероятность отказа оборудования 5%. Входные данные будут отличаться у каждого расписания.

После запуска приложения в главном окне заполнялись данные для 3-х операционного расписания (рисунок 3.4).

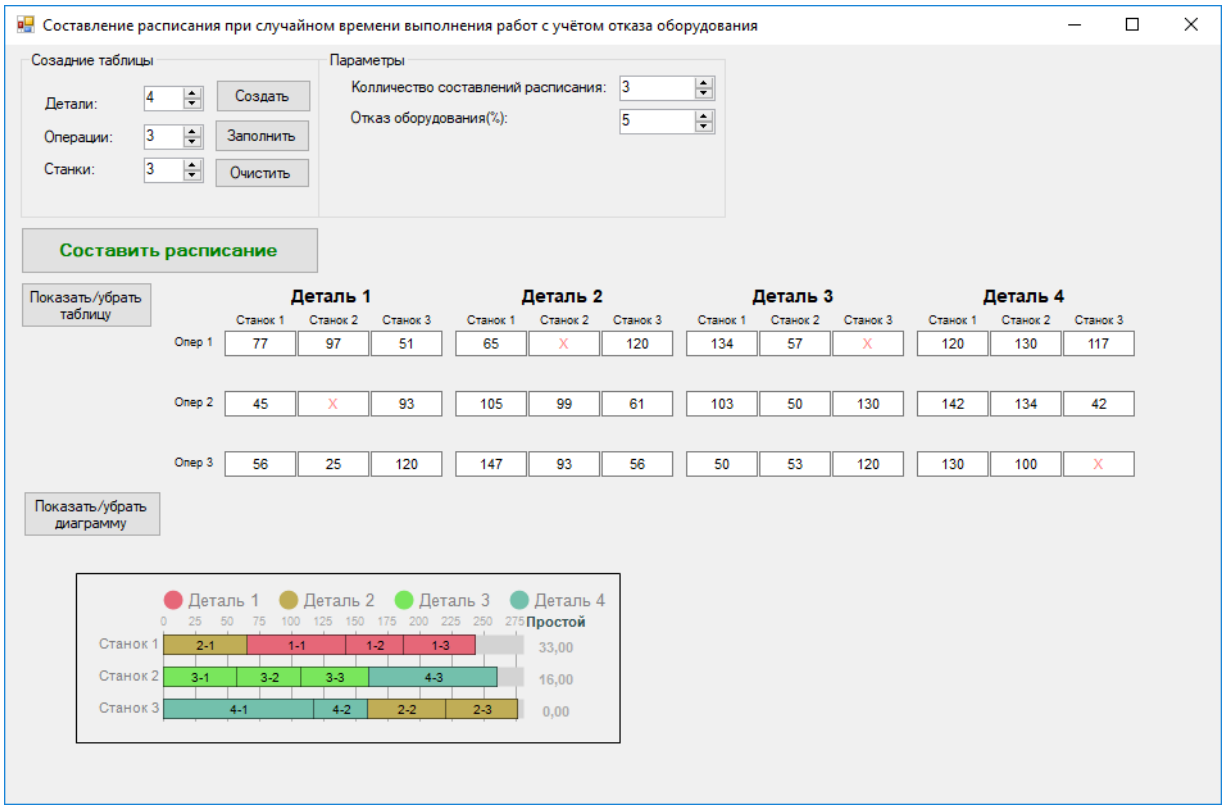

Рисунок 3.4 – Программно-полученное расписание для 3-х операций

Из диаграммы видно, что полученное расписание для 3-х операций, имеет минимальный простой по времени, далее было получено расписание для 4-х операций (рисунок 3.5).

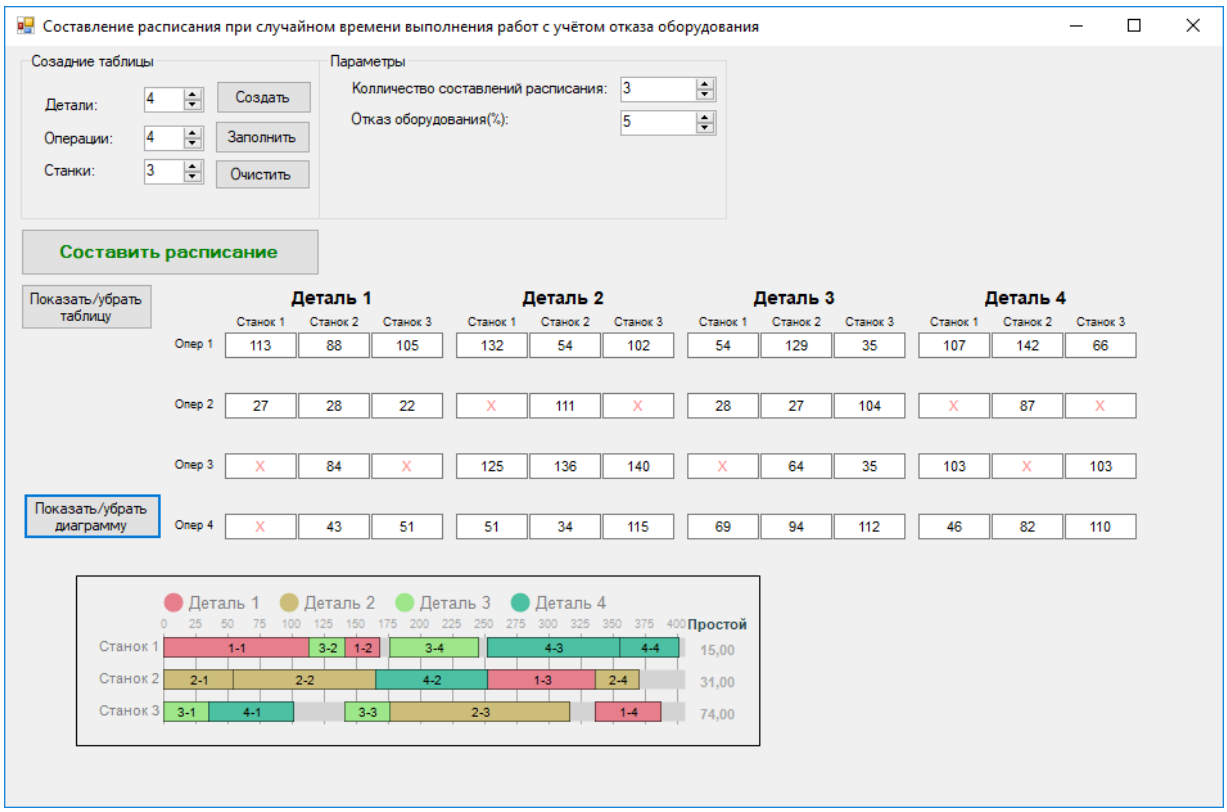

Рисунок 3.5 – Программно-полученное расписание для 4-х операций

Полученное расписание для 4-х операций существенно отличается, общий простой по времени становится больше. Составление расписания по предложенному алгоритму, имеют время окончания последней операции близкое к минимально возможному. Однако, полученное 3-х операционное расписание близко к оптимальному.

Вывод

В результате было разработано все приложение целиком, а также пользовательский интерфейс. Приложение состоит из одного окна. В котором находятся основные параметры расписания и таблицы для заполнения времени работ, а также диаграмма с конечным расписанием.

По результатам проведенного программного эксперимента была подтверждена работоспособность системы и установлено при каком количестве операций программа выдает результат близкий к оптимальному. Обнаруженные в ходе промежуточных экспериментов ошибки устранялись в течении всего периода разработки.

ЗАКЛЮЧЕНИЕ

Задача составления расписания, это задача составить наилучшее, с точки зрения выбранных целей и ограничений, расписания в рамках принятой модели.

В современных условиях конкуренции неоспоримым фактом является огромные усилия на снижение себестоимости продукции. Также у производителя желание автоматизировать составление расписание как на долгосрочный период, так и на краткосрочный период, наряду с логистикой, модернизацией производства и обучением персонала. В связи с этим передо мной была поставлена задача составления расписания для выпуска продукции предприятием.

Анализ поставленной задачи позволяет сделать вывод о ее *NP*-трудности. Для решения поставленной задачи был применен модифицированный алгоритм диспетчеризации. Использование приемлемых, для задачи, эвристик позволяет получить необходимый результат, т.е. эвристические алгоритмы позволяют получить "хорошее" решение за приемлемое время, однако анализ полученного решения, в частности его численная оценка достаточно затруднительны.

При выполнении дипломной работы были получены перечисленные ниже теоретические и практические результаты.

1. Предложена методика построения системы расписания, которая опирается на два принципиальных положения:

– эффективного алгоритма, которое позволит получить оптимальное расписание за приемлемое время не существует;

– важно учитывать производственные процессы, их анализ позволяет определить применимость различных эвристик.

2. В качестве основного алгоритма был выбран однократный алгоритм диспетчеризации. Его выбор обусловлен следующими причинами:

– возможность применения комбинированных эвристик;

– получение расписания за приемлемое время.

3. Разработан ряд процедур, позволяющих реализовать построение производственного расписания указанного вида на вычислительной технике, в условиях ограниченности времени. При этом получены следующие результаты:

– описана методика построения «базового» расписания. При этом под «базовым», понимаем расписание, получаемое простым алгоритмом диспетчеризации;

– на основе базового алгоритма построена система назначения приоритетов операций, основанная на комбинации эвристик, позволяющая классифицировать ранг операции на каждом шаге итерации. Также задан итерационный процесс получения некоторого числа различных расписаний, обеспечивающий поиск локального максимума показателя качества по предложенному критерию.

4. Алгоритмическая база имеет некоторые ограничения. Часть этих ограничений не имеет принципиального характера и при построении модели либо

не учитывается, например, наличие объемных ресурсов, либо упрощается, например, время переналадки оборудования.

5. Эффективность работы предложенных алгоритмов проверена на ряде тестовых и реальных задач. На основе этих алгоритмов построена система составления расписания.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Романова, А.А. Разработка точных и приближенных алгоритмов построения расписаний для производственных систем: дис. … канд. физико-матем. наук / А. А. Романова. – Омск: изд-во ОмГУ., 2006. – 113 с.
2. Арендателева, С. И. Математическая модель и алгоритмы составления расписаний и прогнозирования производства на малом предприятии / С. И. Арендателева. – Великий Новгород, 2010. – 169 с.
3. Лугуев, Т. С. Теоретико-графовые модели и методы составления расписаний без прерываний: дис. … канд. физико-матем. наук / Т. С. Лугуев. – Астрахань: изд-во ДГУ, 2011. – 141 с.
4. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. Optimization and approximation in deterministic sequencing and scheduling: a survey // Ann. Discrete Math, 1979. – v.5. – 287–326 p.
5. Гэри, М. Вычислительные машины и трудно решаемые задачи / М. Гэри, Д. Джонсон; пер. с англ. – М.: Мир, 1982. – 416 с.
6. Тимковский, В.Г. Дискретная математика в мире станков и деталей. / В.Г. Тимковский. – М.: Наука, 1992. – 145 с.
7. Тимковский, В.Г. К сложности составления расписаний произвольной системы // Изв. АН СССР. Техн. кибернетика. – 1985. – №3. – С. 102-109.
8. Кузин, Б.И. Математические модели в оперативном управлении и организации дискретного производства / Б.И. Кузин, Л.Ю. Норинский. СПб.: Изд-во СПбГТУ, 2001. – 233 с.
9. Тимковский, В.Г. Приближенное решение задачи составления расписания циклической системы // Экономика и математические методы – 1986. – №2. – С. 171-174.
10. Федоров, А.М. Реализация функции планирования и MES-системах / А.М. Федоров. – М: МАИ, 2007. – 220 с.
11. Кормен, Т. Алгоритмы: построение и анализ. / Т. Кормен, Ч. Лейзерсон, Р. Ривест. — М.: МЦНМО, 2001. – 734 с.
12. Арендателева, С. И. Методика планирования производственной системы в информационном пространстве программы "1С предприятие" / С. И. Арендателева // Вестник Тверского государственного университета. Серия «Прикладная математика». 2010. – № 2 (17). – С. 97-109.
13. Конвей, Р.В. Теория расписаний. / Р.В. Конвей, В.Л. Максвелл, Л.В. Миллер – М.: Наука, 1975. – 360 с.
14. Теория расписаний и вычислительные машины / под ред. Э.Г. Коффмана. – М.: Наука, 1984. – 336 с.
15. Лившиц, Э.М. Минимизация максимального штрафа в задаче одного станка. / Э.М. Лившиц, – М.: Наука, 1977. – 112 с.

16. Шопин, А.Г., Михайлин, С.А. Инструменты для построения MES систем // Информатизация и системы управления в промышленности. – 2005. – №6 – С. 102-114.
17. Голенко, Д. И. Статистические модели в управлении производством / Д.И. Голенко. – М.: Статистика, 1973. – 368 с.
18. Щепин, Е.В. Теория расписаний / Е.В. Щепин. – М.: Школа Яндекса по анализу данных, 2007. – 56 с.
19. Танаев, В.С. Теория расписаний. Одностадийные системы. / В.С. Танаев, В.С. Гордон, Я.М. Шафранский. – М.: Наука. Гл. ред. физ.-мат. лит., 1989. – 384 с.
20. Евтушенко, Ю. Г. Методы решения экстремальных задач и их применение в системах оптимизации / Ю.Г.Евтушенко. М.: Наука, 1982. –196 с.
21. Леньшин, В.Н., Куминов, В.В. Информатизация производственных процессов – путь к эффективному предприятию // МКА. – 2002. – №3. – С. 93-96.
22. Gantt, H.L. ASME Transactions / H.L. Gantt. – Pa: Hive Publishing Company, 1963. – 1322-1336 p.
23. Amir A., Kaplan E.H. Relocation problems are hard // International Journal of Computer Mathematics – 1988. – Vol. 25 – 101-110p.
24. Story A.E. Computational Experience with Integer Programming for Job-Shop Scheduling / A.E. Story, H.M. Wagner Englewood Cliffs, N.J.: Prentice-Hall, 1963. – 14p.
25. Giglio R.J. Approximate Solutions to the Three-Machine Scheduling Problems / R.J. Giglio, H.M. Wagner // Oper. Res. 1964. – V.12. №2. – 305-324 p.
26. Bellman R. Mathematical Aspects of Scheduling Theory / R. Bellman // J. Soc. Indust. and Appl. Math. 1956. – V.4, №3. – 168-205 p.
27. Paraskevopoulos D., Karakitsos E., Rustem B. Robast capasity, planning under uncertanty // Management scince Providence, 1990. – Vol. 37,№ 7. – 787-801 p.
28. Kirkwood G.W. Estimating the impact of uncertainty on a deterministic multiattribute evaluation // Management science Providence, 1992. – Vol. 38, № 6. – 819 - 831 p.
29. Johnson S.M. Optimal Two and Three Stage Production Schedules with Set Up Times Included / S.M. Johnson // Nav. Res. Log. Quart. 1954. – V.l, №1. – 61-68 p.
30. Bellman R. Some Combinatorial Problems Arising in the Theory of Multistage Processes / R. Bellman, O. Gross // J. Soc. 1.dust. and Appl. Math. – 1954. –V.2, №3. – 175 -183 p.

ПРИЛОЖЕНИЕ 1

П1.1 Текст программы Algorithm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

namespace Scheduling
{
    class GeneticMachine
    {
        int jobnum, procnum, macnum, krlength = 0;
        bool refresh;
        Schedule[] population;
        Random rnd = new Random();
        private Schedule best;
        double progress = 0;
        int produced = 0;
        bool stopped;
        int mutOdd;
        int groupSize;
        int minTimeOdd;
        int popSize;
        AsyncOperation aop = AsyncOperationManager.CreateOperation(null);
        COTypes crossOver;
        MutationTypes mutationTypes;
        SelectionTypes selectionType;
        public event EventHandler BestValueChanged;
        public event EventHandler ProgressChanged;
        int nothingFound;

        public GeneticMachine(int jobs, int procs, int macs, int popsize)
        {
            this.jobnum = jobs;
            this.procnum = procs;
            this.macnum = macs;
            krlength = jobs * procs;
            mutOdd = 1;
            minTimeOdd = 0;
            groupSize = 5;
            population = new Schedule[popsize];
            popSize = popsize;
            crossOver = COTypes.Uniform;
            mutationTypes = MutationTypes.ExchangeValues;
            selectionType = SelectionTypes.Tournament;
            best = null;
            nothingFound = 0;
        }
        void CreateInitialPopulation()
        {
            for (int i = 0; i < population.Length; i++)
            {
                Schedule ciz = createNewSchedule();
                population[i] = ciz;
                produced++;

                Progress = double.Parse((produced * 100d / 10000000).ToString("0.00"));
                if (stopped)
```

```
                break;
            }
        }
        public void Start()
        {
            CreateInitialPopulation();
            Procedure();
        }
        void Procedure()
        {
            Task t1 = new Task(() =>
            {
                stopped = false;
                while (produced < 10000000)
                {
                    int[] nums = doSelection();

                    Schedule mother = population[nums[0]];
                    Schedule father = population[nums[1]];

                    Schedule child1 = doCrossover(mother, father);
                    Schedule child2 = doCrossover(father, mother);

                    doMutation(child1);
                    doMutation(child2);

                    population[nums[nums.Length - 1]] = child1;
                    population[nums[nums.Length - 2]] = child2;
                    checkBestValueChanged(child1);
                    checkBestValueChanged(child2);
                    produced++;
                    nothingFound++;
                    if (nothingFound > 300000 && refresh)
                    {
                        addNewChromosomes(popSize / 10);
                        nothingFound = 0;
                    }
                    Progress = double.Parse((produced * 100d / 10000000).ToString("0.00"));
                    if (stopped)
                        break;
                }
            });
            t1.Start();
        }
        void addNewChromosomes(int n)
        {
            int[] nums = new int[n];
            for (int i = 0; i < nums.Length; i++)
            {
                nums[i] = rnd.Next(0, population.Length);
                Schedule created = createNewSchedule();
                population[nums[nums.Length - 1 - i]] = created;
                checkBestValueChanged(created);
                produced++;
            }

            }
Schedule createNewSchedule()
        {
            int[] jobSchedule = new int[krlength];
            int[] macSchedule = new int[krlength];
            int[] used = new int[jobnum];
            bool[] assigned = new bool[krlength];
```

```
    List<int> rem = Enumerable.Range(0, jobnum).ToList();
    for (int j = 0; j < krlength; j++)
    {
        int selJob = rem[rnd.Next(0, rem.Count)];
        jobSchedule[j] = selJob;
        if (rnd.Next(1, 101) <= minTimeOdd)
        {
            int minmac = Data.GetMinTimeMacForJP(selJob, used[selJob]);
            int pos = selJob * procnum + used[selJob];
            macSchedule[pos] = minmac;
            assigned[pos] = true;
        }
        else
        {
            if (!assigned[j])
                macSchedule[j] = rnd.Next(0, macnum);
        }
        used[selJob]++;
        if (used[selJob] == procnum)
            rem.Remove(selJob);
    }

   return new Schedule(jobSchedule, macSchedule, jobnum, procnum, macnum);
}

int[] doSelection()
{
    int[] nums = null;
    switch (selectionType)
    {
        case SelectionTypes.Tournament:
            nums = tourSelection(groupSize);
            break;

    }
    return nums;
}

public void doMutation(Schedule child)
{
    MutationTypes temp = mutationTypes;
    if(rnd.Next(1, 101) <= mutOdd)
    switch (temp)
    {
        case MutationTypes.ExchangeValues:
            ExchangeValuesMutation(child);
            break;
    }
}

Schedule doCrossover(Schedule mother, Schedule father)
{
    COTypes temp = crossOver;
    Schedule child = null;
    switch (temp)
    {
        case COTypes.Uniform:
            child = UniformCO(mother, father);
            child.Repair();
            break;

    }
    return child;
```

```
        }
        public void Stop()
        {
            stopped = true;
        }

        void checkBestValueChanged(Schedule tour)
        {
            if (best == null || best.FitValue > tour.FitValue)
            {
                best = tour;
                aop.Post(new System.Threading.SendOrPostCallback(delegate
                {
                    if (BestValueChanged != null)
                        BestValueChanged(this, EventArgs.Empty);
                }), null);
                nothingFound = 0;
            }
        }
        void fireProgressEvent()
        {
            aop.Post(new System.Threading.SendOrPostCallback(delegate
            {
                if (ProgressChanged != null)
                    ProgressChanged(this, EventArgs.Empty);
            }), null);
        }
        int[] tourSelection(int groupSize)
        {
            int[] nums = new int[groupSize];

            for (int i = 0; i < nums.Length; i++)
            {
                nums[i] = rnd.Next(population.Length - 1);
            }
            sortArray(nums);
            return nums;
        }
        void sortArray(int[] nums)
        {
            bool sorted = true;
            while (sorted)
            {
                sorted = false;

                for (int i = 0; i < nums.Length - 1; i++)
                {
                    if (population[nums[i]].FitValue > population[nums[i + 1]].FitValue)
                    {
                        int temp = nums[i];
                        nums[i] = nums[i + 1];
                        nums[i + 1] = temp;
                        sorted = true;
                    }
                }
            }
            }
Schedule UniformCO(Schedule mother, Schedule father)
        {
            int N = krlength;
            int[] macs = mother.MacSchedule.Clone() as int[];
            int[] jobs = mother.JobSchedule.Clone() as int[];
            for (int i = 0; i < N; i++)
```

```
    {
        if (rnd.Next(0, 2) == 0)
        {
            macs[i] = father.MacSchedule[i];
        }
        if (rnd.Next(0, 2) == 0)
        {
            jobs[i] = father.JobSchedule[i];
        }
    }
    return new Schedule(jobs, macs, jobnum, procnum, macnum);
}
void ExchangeValuesMutation(Schedule sch)
{
    int N = krlength;
    int x1 = rnd.Next(0, N);
    int x2 = rnd.Next(0, N);

    int temp = sch.JobSchedule[x1];
    sch.JobSchedule[x1] = sch.JobSchedule[x2];
    sch.JobSchedule[x2] = temp;

    x1 = rnd.Next(0, krlength);
    x2 = rnd.Next(0, krlength);

    int macTemp = sch.MacSchedule[x1];
    sch.MacSchedule[x1] = sch.MacSchedule[x2];
    sch.MacSchedule[x2] = macTemp;
}

public bool Refresh
{
    get { return refresh; }
    set { refresh = value; }
}
public SelectionTypes SelectionType
{
    get { return selectionType; }
    set { selectionType = value; }
}

public int MinTimeOdd
{
    get { return minTimeOdd; }
    set { minTimeOdd = value; }
}

public int GroupSize
{
    get { return groupSize; }
    set { groupSize = value; }
}

public int MutOdd
{
    get { return mutOdd; }
    set { mutOdd = value; }
}

public MutationTypes MutationTypes
{
    get { return mutationTypes; }
    set { mutationTypes = value; }
```

```
        }

        public COTypes CrossOver
        {
            get { return crossOver; }
            set { crossOver = value; }
        }

        public bool Stopped
        {
            get
            {
                return stopped;
            }
        }
        public Schedule Best
        {
            get { return best; }
            set { best = value; }
        }
        public double Progress
        {
            get
            {
                return progress;
            }
            set
            {
                if (progress != value)
                    fireProgressEvent();
                progress = value;
            }
        }



        void refreshPopulation()
        {
            for (int i = 0; i < popSize; i++)
            {
                Schedule ciz = createNewSchedule();
                population[i] = ciz;
                checkBestValueChanged(ciz);
            }
        }
    }

    public static class Data
    {
        public static float[, ,] DataTable;
        public static int GetMinTimeMacForJP(int job, int proc)
        {
            int result = -1;
            float min = -1;
            for (int i = 0; i < DataTable.GetLength(2) - 1; i++)
            {
                float curr = Math.Min(min == -1 ? Data.DataTable[job, proc, i] : min,
Data.DataTable[job, proc, i + 1]);
                if (min == -1)
                {
                    result = (curr == DataTable[job, proc, i] ? i : i + 1);
                    min = curr;
                }
```

```
                if (curr < min)
                {
                    result = i + 1;
                    min = curr;
                }
            }
            return result;
        }
    }
    public enum COTypes
    {

        Uniform = 2,

    }
    public enum MutationTypes
    {
        ExchangeValues = 0,

    }
    public enum SelectionTypes
    {
        Tournament = 0,

    }
    }
```

П1.2 Текст программы Colors.cs

```
using System;
using System.Drawing;

namespace Scheduling
{
    public static class Colors
    {
        static Random rnd = new Random();
        public static Color[] JobColors;
        public static void GenerateRandomHSV(int jobs)
        {
            JobColors = new Color[jobs];
            double a = 0.8d / (jobs + 1);

            for (int i = 0; i < jobs; i++)
            {
                double h = a * (i + 1) + 0.2d;
                double s = rnd.Next(0, 5) * 0.05 + 0.4;
                double v = rnd.Next(0, 5) * 0.05 + 0.7;
                JobColors[i] = GenerateHSVColor(h, s, v);
            }
        }
        public static Color GenerateHSVColor(double h, double s, double v)
        {
            h += 0.61803398;
            h = h > 1 ? h - 1 : h;
            int hi = (int)(h * 6);
            double f = h * 6 - hi;
            double p = v * (1 - s);
            double q = v * (1 - f * s);
            double t = v * (1 - (1 - f) * s);
            double r = 0, g = 0, b = 0;
            if (hi == 0)
            {
                r = v;
```

```
                g = t;
                b = p;
            }
            if (hi == 1)
            {
                r = q;
                g = v;
                b = p;
            }
            else if (hi == 2)
            {
                r = p;
                g = v;
                b = t;
            }
            else if (hi == 3)
            {
                r = p;
                g = q;
                b = v;
            }
            else if (hi == 4)
            {
                r = t;
                g = p;
                b = v;
            }
            else if (hi == 5)
            {
                r = v;
                g = p;
                b = q;
            }
            int R = (int)(r * 256);
            int G = (int)(g * 256);
            int B = (int)(b * 256);

            return Color.FromArgb(R, G, B);
        }
    }
}
```

П1.3 Текст программы InputBoxProvider.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Scheduling
{
    class InputBoxProvider
    {
        int space;
        int width;

        int extra;
        public InputBoxProvider()
        {
            space = 1;
            width = 46;

            extra = 10;
        }
```

```
        Panel CreateInputBox(int macs, int procs, int jobIndex)
        {
            int n = macs * procs;
            TextBox[] boxes = new TextBox[n];
            Panel grup = new Panel()
            {
                Width = (width + extra) * macs + (macs - 1) * space,
                Height = width * procs + (procs - 1) * space,
                Name = "boxer"
            };
            int a = 0;
            for (int i = 0; i < procs; i++)
            {
                for (int j = 0; j < macs; j++)
                {
                    boxes[a] = new TextBox()
                    {
                        Font = new Font("Arial", 8f),
                        Location = new Point(j * (width + extra + space), i * (width +
space)),
                        Name = "Деталь " + jobIndex + "Опер " + (i + 1) + "Станок " + (j + 1),
                        Size = new Size(width + extra, width),
                        Multiline = false,
                        TextAlign = HorizontalAlignment.Center
                    };
                    boxes[a].TextChanged += InputBoxProvider_TextChanged;
                    grup.Controls.Add(boxes[a]);
                    a++;
                }
            }
            return grup;
        }
        void InputBoxProvider_TextChanged(object sender, EventArgs e)
        {
            TextBox tb = sender as TextBox;

            if (tb.Text.ToLower() == "x")
            {
                tb.Text = "X";
                tb.ForeColor = Color.FromArgb(255, 122, 122);
                tb.SelectionStart = tb.Text.Length;
            }
            else
                tb.ForeColor = tb.ForeColor = Color.Black;
        }

        Panel CreateMachineBox(int macs)
        {
            Panel grup = new Panel()
            {
                Width = macs * (width + extra) + (macs - 1) * space,
                Height = 15
            };
            for (int i = 0; i < macs; i++)
            {
                Label lbl = new Label()
                {
                    AutoSize = false,
                    Width = width,
                    Height = 15,
                    Text = "Станок " + (i + 1),
                    Location = new Point(i * (width + extra + space) + 6, 0),
                    Font = new Font("Arial", 6.75f),
```

```
            TextAlign = ContentAlignment.MiddleCenter
        };
        grup.Controls.Add(lbl);
    }
    return grup;
}


Panel CreateProcessBox(int procs)
{
    Panel grup = new Panel()
    {
        Width = width,
        Height = procs * width + (procs - 1)
    };
    for (int i = 0; i < procs; i++)
    {
        Label lbl = new Label()
        {
            AutoSize = false,
            Width = width,
            Height = 15,
            Text = "Опер " + (i + 1),
            Location = new Point(0, i * (width + space)),
            Font = new Font("Arial", 6.75f),
            TextAlign = ContentAlignment.MiddleCenter
        };
        grup.Controls.Add(lbl);
    }
    return grup;
}

Label CreateJobTitleLabel(int jobIndex, int macs)
{
    Label lbl = new Label();
    lbl.AutoSize = false;
    lbl.Width = macs * (width + extra) + (macs - 1) * space; //Общая ширина:1px
    lbl.Text = "Деталь " + jobIndex;
    lbl.TextAlign = ContentAlignment.MiddleCenter;
    lbl.Font = new Font("Arial", 10, FontStyle.Bold);
    return lbl;
}

Panel CreateJobBox(int macs, int procs, int jobIndex)
{
    Label title = CreateJobTitleLabel(jobIndex, macs);
    Panel macBox = CreateMachineBox(macs);
    Panel boxes = CreateInputBox(macs, procs, jobIndex);
    Panel container = new Panel()
    {
        Width = macBox.Width,
        Height = title.Height + macBox.Height + boxes.Height,
        Name = "Деталь " + jobIndex
    };
    title.Location = new Point(0, 0);
    macBox.Location = new Point(0, title.Bottom);
    boxes.Location = new Point(0, macBox.Bottom);

    container.Controls.Add(title);
    container.Controls.Add(macBox);
    container.Controls.Add(boxes);
    return container;
}
```

```
        public Panel CreateJobBoxes(int mac, int proc, int job)
        {
            Panel procBox = CreateProcessBox(proc);
            procBox.Location = new Point(0, 40);
            Panel container = new Panel();
            container.BackColor = Color.Transparent;
            container.Controls.Add(procBox);
            int x = 0;
            for (int i = 0; i < job; i++)
            {
                Panel jobBox = CreateJobBox(mac, proc, i + 1);
                jobBox.Name = "Деталь " + (i + 1);
                jobBox.Location = new Point(procBox.Width + x + (i == 0 ? 0 : 10), 0);
                container.Controls.Add(jobBox);
                if (i == job - 1)
                {
                    container.Height = jobBox.Bottom + 30;
                }
                x = jobBox.Right - procBox.Width;
            }
            return container;
        }


    }
}
```

П1.4 Текст программы Schedule.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace Scheduling
{
    public class Schedule
    {
        private int[] jobSchedule;
        private int[] macSchedule;
        int j, p, m;
        float[,] startTimes, finishTimes;
        float[] idleTimes;
        float totalIdleTime;
        bool makeSpanCalculated, idleCalculated;
        float makeSpan;

        public Schedule(int[] js, int[] ms, int j, int p, int m)
        {
            this.j = j;
            this.p = p;
            this.m = m;
            jobSchedule = js;
            macSchedule = ms;

            makeSpanCalculated = false;
            idleCalculated = false;
            makeSpan = 0;
            startTimes = new float[j, p];
            finishTimes = new float[j, p];
            idleTimes = new float[m];
```

```
        }

        public float MakeSpan()
        {
            if (makeSpanCalculated)
                return makeSpan;

            int[] compProcs = new int[j];
            float[] totalTime = new float[m];

            int?[] lastJob = new int?[m];
            int?[] lastProc = new int?[m];

            for (int i = 0; i < jobSchedule.Length; i++)
            {
                int job = jobSchedule[i];
                int proc = compProcs[job];
                int a = job * p + proc;
                int mac = macSchedule[a];
                int oldProc = compProcs[job] > 0 ? compProcs[job] - 1 : -1;

                startTimes[job, proc] = Math.Max(totalTime[mac],
                                                 oldProc == -1 ? 0 : finishTimes[job,
oldProc]);
                idleTimes[mac] += startTimes[job, proc] - totalTime[mac];
                lastJob[mac] = job;
                lastProc[mac] = proc;

                finishTimes[job, proc] = startTimes[job, proc] + Data.DataTable[job, proc,
mac];

                totalTime[mac] = finishTimes[job, proc];
                compProcs[job]++;
            }

            float result = -1;
            for (int i = 0; i < j; i++)
            {
                result = Math.Max(finishTimes[i, p - 1], result);
            }
            makeSpan = result;
            for (int i = 0; i < m; i++)
            {
                if (lastProc[i] != null)
                    idleTimes[i] += makeSpan - finishTimes[(int)lastJob[i], (int)lastProc[i]];
                else
                    idleTimes[i] = makeSpan;
                if (i == m - 1)
                    idleTimes[i] = float.Parse(idleTimes[i].ToString("0.00"));

            }


            makeSpanCalculated = true;
            return makeSpan;
        }

        public void DrawSchedule(Graphics grp, Control panel)
        {
            if (!makeSpanCalculated)
                MakeSpan();

            grp.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.HighQuality;
```

```
            int mac = m;
            int proc = p;
            int job = j;
            int horSpace = 70;
            int vertSpace = 50;
            int delWidth = 0;


            for (int i = 0; i < job; i++)
            {
                Rectangle rect = new Rectangle(90 * i + horSpace, 15, 15, 15);
                string text = "Деталь " + (i + 1);
                Point tp = new Point(rect.Right + 3, rect.Y);
                using (Font font = new Font("Arial", 10))
                using (SolidBrush brush = new SolidBrush(Colors.JobColors[i]))
                {
                    grp.FillEllipse(brush, rect);
                    grp.DrawString(text, font, Brushes.Gray, tp);
                }
                if (i == job - 1)
                    delWidth = tp.X + 20;
            }


            panel.Height = (mac * 25 + vertSpace) + 15;
            panel.Width = (int)Math.Max(delWidth, makeSpan + horSpace + 15) + 50;
            grp.DrawRectangle(Pens.Black, 2, 2, panel.Width - 4, panel.Height - 8);

            int idWidth = 0;
            int num = (int)((makeSpan + 5) / 25);
            for (int i = 0; i <= num; i++)
            {
                Point p1 = new Point(25 * i + horSpace, vertSpace - 3);
                Point p2 = new Point(25 * i + horSpace, vertSpace + (int)((mac - 0.5) * 25) +
5);
                using (Font font = new Font("Arial", 7))
                {
                    grp.DrawLine(Pens.DarkGray, p1, p2);
                    string text = (25 * i).ToString();
                    Size size = TextRenderer.MeasureText(text, font);
                    Point tp = new Point(p1.X - size.Width / 2 + 2, p1.Y - size.Height);
                    grp.DrawString(text, font, Brushes.DarkGray, tp);

                }
                }
            PointF itPo = Point.Empty;
            RectangleF[] backRects = new RectangleF[mac];
            for (int i = 0; i < mac; i++)
            {
                backRects[i] = new RectangleF(horSpace, vertSpace + 25 * i, makeSpan + 5, 15);
                grp.FillRectangle(Brushes.LightGray, backRects[i]);
                using (Font font = new Font("Arial", 8, FontStyle.Bold))
                {
                    if (i == 0)
                    {
                        Size idSize = TextRenderer.MeasureText("Простой", font);
                        itPo = new PointF(backRects[i].X + backRects[i].Width, vertSpace - 3 -
idSize.Height);
                        grp.DrawString("Простой", font, Brushes.DarkSlateGray, itPo);
                        idWidth = TextRenderer.MeasureText("Простой", font).Width;
                    }
                    string idle = idleTimes[i].ToString("0.00");
                    Size size = TextRenderer.MeasureText(idle, font);
```

```
                    grp.DrawString(idle, font, Brushes.DarkGray, itPo.X + idWidth / 2 -
size.Width / 2,
                                                                    backRects[i].Y +
backRects[i].Height / 2 - size.Height / 2 + 2);
                }
            }

            for (int i = 0; i < mac; i++)
            {
                using (Font font = new Font("Arial", 8))
                {
                    string text = "Станок " + (i + 1);
                    Size size = TextRenderer.MeasureText(text, font);
                    PointF tp = new PointF(backRects[i].X - size.Width, backRects[i].Y);
                    grp.DrawString(text, font, Brushes.Gray, tp);

                }
            }

            for (int i = 0; i < job; i++)
            {
                for (int b = 0; b < proc; b++)
                {
                    int tempJob = i;
                    int tempProc = b;
                    int o = i * p + b;
                    int tempMac = macSchedule[o];

                    float x = startTimes[i, b] + horSpace;
                    float y = backRects[tempMac].Y;
                    float height = 15;
                    float time = Data.DataTable[i, b, tempMac];

                    PointF ps1 = new PointF(x, y + 15);
                    PointF ps2 = new PointF(x, ps1.Y - height);
                    PointF ps3 = new PointF(ps2.X + time, ps2.Y);
                    PointF ps4 = new PointF(ps3.X, ps3.Y + height);

                    GraphicsPath path = new GraphicsPath();
                    path.AddLine(ps1, ps2);
                    path.AddLine(ps2, ps3);
                    path.AddLine(ps3, ps4);
                    path.CloseAllFigures();

                    RectangleF bar = new RectangleF(x, y, ps3.X - ps1.X, height);
                    using (SolidBrush brush = new SolidBrush(Colors.JobColors[i]))
                    {
                        grp.DrawPath(Pens.Black, path);
                        grp.FillPath(brush, path);
                    }
                    using (Font font = new Font("Arial", 7.5f))
                    {
                        string text = (i + 1) + "-" + (b + 1);
                        SizeF size = TextRenderer.MeasureText(text, font);
                        if (size.Width > bar.Width)
                        {
                            text = (b + 1).ToString();
                            size = TextRenderer.MeasureText(text, font);
                        }
                        PointF tp = new PointF(bar.X + bar.Width / 2 - size.Width / 2 + 3,
bar.Y + height / 2 - size.Height / 2 + 1);
                        grp.DrawString(text, font, Brushes.Black, tp);
                    }
```

```
        }
    }
}
Random rnd = new Random();
public void Repair()
{
    int num = p;
    int[] used = new int[j];
    List<int> pos = new List<int>();
    List<int> jobs = new List<int>();
    for (int i = 0; i < p * j; i++)
    {
        used[jobSchedule[i]]++;
        if (used[jobSchedule[i]] > num)
            pos.Add(i);
    }
    for (int i = 0; i < used.Length; i++)
    {
        if (used[i] < num)
        {
            for (int t = 0; t < num - used[i]; t++)
            {
                jobs.Add(i);
            }
        }
    }
    for (int i = 0; i < pos.Count; i++)
    {
        int t = rnd.Next(0, jobs.Count);
        jobSchedule[pos[i]] = jobs[t];
        jobs.RemoveAt(t);
    }
}

public float FitValue
{
    get
    {
        return MakeSpan();
    }
}
public int[] MacSchedule
{
    get { return macSchedule; }
    set { macSchedule = value; }
}
public int[] JobSchedule
{
    get { return jobSchedule; }
    set { jobSchedule = value; }
}
public float TotalIdleTime
{
    get
    {
        MakeSpan();
        if (idleCalculated)
            return totalIdleTime;
        float total = 0;
        for (int i = 0; i < m; i++)
        {
            total += idleTimes[i];
        }
```

```
                totalIdleTime = total;
                idleCalculated = true;
                return totalIdleTime;
            }
        }
        public string JobsToString()
        {
            string text = "";
            for (int i = 0; i < j * p; i++)
            {
                text += (jobSchedule[i]+1) + "";
            }
            return text;
        }
        public string MacToString()
        {
            string text = "";
            for (int i = 0; i < j * p; i++)
            {
                text += (macSchedule[i] + 1) + "";
            }
            return text;
        }

    }
    }
```