В работе рассмотрены вопросы, связанные с построением математической модели вычислительных систем реального времени и планированием вычислений в таких системах. Разработаны алгоритмы построения многопроцессорных расписаний: 1) с прерываниями и переключениями с одного процессора на другой (сравнительный анализ точных и эвристических алгоритмов; получение необходимых и достаточных условий существования допустимого расписания в виде аналитических ограничений на параметры системы); 2) без прерываний и переключений (точные и приближенные алгоритмы; разработка метода агрегирования); 3) в случае, когда заданы директивные интервалы, а длительности выполнения работ линейно зависят от количества выделенного им дополнительного ресурса; 4) в случае, когда требования на выполнение работ поступают циклически с заданными периодами; 5) при наличии неопределенных факторов. Разработаны алгоритмы организации контроля в системах реального времени (оптимизация структуры подсистемы контроля). Книга предназначена для специалистов в области проектирования систем реального времени и теории расписаний, а также студентов и аспирантов соответствующих специальностей.

Меран Фуругян

Алгоритмы планирования вычислений и синтеза систем реального времени

Разработка и сравнительный анализ

Меран Фуругян

Фуругян Меран Габибуллаевич, кандидат физико-математических наук, заведующий сектором Вычислительного центра им. А.А. Дородницына РАН, доцент МГУ им. М.В. Ломоносова и МФТИ, Москва. Область научных интересов: разработка систем реального времени, теория расписаний, теория игр и исследования операций.

978-3-8484-8847-6

LAP LAMBERT Academic Publishing

Алгоритмы планирования вычислений в СРВ

Меран Фуругян

Меран Фуругян

Алгоритмы планирования вычислений и синтеза систем реального времени

Меран Фуругян

Алгоритмы планирования вычислений и синтеза систем реального времени

Разработка и сравнительный анализ

LAP LAMBERT Academic Publishing

Impressum/Imprint (nur für Deutschland/only for Germany)
Bibliografische Information der Deutschen Nationalbibliothek: Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Coverbild: www.ingimage.com

Verlag: LAP LAMBERT Academic Publishing GmbH & Co. KG
Heinrich-Böcking-Str. 6-8, 66121 Saarbrücken, Deutschland
Telefon +49 681 3720-310, Telefax +49 681 3720-3109
Email: info@lap-publishing.com

Herstellung in Deutschland:
Schaltungsdienst Lange o.H.G., Berlin
Books on Demand GmbH, Norderstedt
Reha GmbH, Saarbrücken
Amazon Distribution GmbH, Leipzig
ISBN: 978-3-8484-8847-6

Только для России и стран СНГ
Библиографическая информация, изданная Немецкой Национальной Библиотекой. Немецкая Национальная Библиотека включает данную публикацию в Немецкий Книжный Каталог; с подробными библиографическими данными можно ознакомиться в Интернете по адресу http://dnb.d-nb.de.

Изображение на обложке предоставлено: www.ingimage.com

Издатель: LAP LAMBERT Academic Publishing GmbH & Co. KG
Heinrich-Böcking-Str. 6-8, 66121 Saarbrücken, Germany
Телефон +49 681 3720-310, Факс +49 681 3720-3109
Email: info@lap-publishing.com

Напечатано в России
ISBN: 978-3-8484-8847-6

Содержание

Введение

При разработке и эксплуатации сложных технических объектов широко применяются специализированные вычислительные системы, функционирующие в непосредственном взаимодействии с внешней средой. За строго ограниченный сверху интервал времени система должна вернуть среде результаты обработки в виде управляющих воздействий или в виде сообщений пользователю. Предметом исследования данной работы являются системы *жесткого реального времени.* Это такие системы, в которых некоторым заданиям сопоставляются директивные сроки, не подлежащие нарушению. Так как время является одним из основных параметров в системе жесткого реального времени, то для обеспечения работоспособности и надежности таких систем необходимо иметь заранее рассчитанное расписание работы всей системы. В данной работе рассматриваются эффективные алгоритмы составления расписаний для некоторых видов многопроцессорных систем жесткого реального времени, задача синтеза таких систем, а также алгоритмы организации контроля.

Рассматриваемый класс задач имеет, помимо чисто научной, большую практическую важность. Потребность в быстрых алгоритмах, составляющих многопроцессорные расписания, часто возникает в задачах распределенных вычислений в реальном времени и задачах оперативного управления на основе обработки и анализа поступающих в реальном времени данных. В качестве иллюстрации широкой практической распространенности систем реального времени и важности эффективных алгоритмов для их расчета и управления можно привести следующие примеры.

1. Современные системы противоракетной обороны представляют собой системы реального времени, непрерывно обрабатывающие поступающие данные о движении объектов в околоземном космическом пространстве. Значение корректной и эффективной работы таких систем трудно переоценить.

2. Системы управления ядерными реакторами на АЭС получают в режиме реального времени данные с множества датчиков и должны на их основе оперативно осуществлять управляющие воздействия на

реактор. В случае, если какие-либо работы этой системы реального времени не будут выполнены в директивный срок, ядерная реакция может выйти из-под контроля и привести к крайне нежелательным последствиям вплоть до взрыва реактора.

3. В аналитических центрах при правительствах развитых стран системы реального времени используются для мониторинга и анализа непрерывно поступающей из различных точек экономической или экологической информации. Такие системы должны эффективно обрабатывать огромные массивы данных и, исходя из них, оперативно оповещать о каких-то замеченных проблемах на ранних стадиях их возникновения.

4. При испытаниях летательных аппаратов и других сложных технических объектов большое значение приобретает получение и оперативная обработка вычислительной системой реального времени периодически поступающей информации о состоянии различных узлов объекта.

5. Космический аппарат является сложным техническим объектом, большинство систем которого полностью автоматизированы. В течение времени управляемого полета бортовые системы аппарата должны выполнить множество операций к определенным срокам, нарушение которых ведет к потере контроля над аппаратом.

6. В современном аэропорту с большим количеством взлетно-посадочных полос постоянно принимаются решения о назначении тех или иных самолетов на различные полосы и порядке, в котором этим самолетам следует взлетать и садиться. Более того, следует также постоянно резервировать свободные полосы и запасы времени на используемых полосах на случай всевозможных нештатных ситуаций, когда неисправный самолет блокирует некоторую взлетно-посадочную полосу.

Необходимо отметить, что корректность систем реального времени зависит не только от правильности результатов ее вычислений, но и от времени, за которое эти результаты были получены. Составление расписаний реального времени – важная часть подобных систем, так как проектировщик системы должен быть уверен в том, что все задания будут исполнены в срок. Одними из основополагающих работ

в области планирования вычислений и многопроцессорных систем, идеи, подходы и основные положения которых использовались в настоящей работе, можно назвать [1], [2], [3] и [4]. Хорошие и обзоры по составлению расписаний в системах реального времени можно найти в [5, 6, 52], а подробное резюме по результатам сравнения различных алгоритмов составления многопроцессорных расписаний приведено в [7].

Согласно общепринятой теории, все алгоритмы составления расписаний делятся на два класса: первые относятся к *диспетчеризации с приоритетным управлением* (priority-driven dispatching), вторые – к *диспетчеризации с управлением при помощи временной шкалы* (timeline-driven dispatching). В алгоритмах диспетчеризации с приоритетным управлением точное время, в которое начнется или завершится какая-либо работа, заранее неизвестно. Время назначения каждой работы зависит от основанного на характеристиках данной работы приоритета относительно других задач, которые необходимо выполнить системе. В свою очередь, в алгоритмах диспетчеризации с управлением при помощи временной шкалы для составления расписания используется только временная шкала. Таким образом, время назначения и выполнения каждой работы известно заранее.

Составление расписаний с прерываниями обычно применяется в системах, в которых невелико время переключения контекста. Примерами систем реального времени, в которых реализованы многопроцессорные расписания с прерываниями, могут служить RT-Mach [8] и LynxOS [9]. Большинство систем реального времени с возможностью прерывания работ используют диспетчеризацию с приоритетным управлением, как и некоторые системы, рассмотренные и предложенные в данной работе. Поэтому остановимся подробнее именно на этом классе алгоритмов.

В алгоритмах диспетчеризации с приоритетным управлением каждой задаче назначается приоритет, и на процессоры назначаются задачи с наивысшими приоритетами среди готовых к запуску задач. Эти алгоритмы предписывают правила назначения приоритетов задачам, а также предоставляют методы проверки возможности составления допустимого расписания.

Все расписания для вычислительных систем принято делить на *статические,* когда информация обо всех назначенных к выполнению системой работах и ограничениях на их выполнение (таких, как директивные интервалы в системах реального времени) известна заранее, и *динамические*, когда изначально не предусмотренные работы могут быть назначены к выполнению системой уже в процессе ее работы. Примером статического расписания может служить обработка информации о каком-либо процессе, поступающая с фиксированного набора датчиков и содержащая заранее известный объем и структуру данных. Примером динамического расписания в системе реального времени может служить алгоритм работы роботов, убирающих загрязненную область заранее неизвестной, возможно, постоянно меняющейся структуры.

Для динамических расписаний получено немного теоретических результатов, в частности, в [22] доказано, что для случая двух и более процессоров, никакой алгоритм построения расписания не может гарантировать построения допустимого расписания, если заранее не известны все директивные интервалы и все вычислительные сложности работ. В данной работе будут рассматриваться только статические расписания, теория которых для систем реального времени на настоящий момент проработана более глубоко. Далее перечислим ее основные известные результаты.

Один из наиболее фундаментальных результатов в теории составления статических расписаний для систем реального времени с прерываниями получили Лю и Лейланд в работе [10], в которой изучались периодически возникающие работы. Они разделили алгоритмы составления расписаний с прерываниями работ на два класса – *статических приоритетов* и *динамических приоритетов*. В алгоритме, относящемся к классу статических приоритетов, приоритет каждой из работ задается тогда, когда эта работа назначается на выполнение и остается неизменным на всех этапах ее выполнения. Напротив, в алгоритме динамических приоритетов приоритет каждой работы может изменяться в процессе выполнения каждого из ее этапов. Для каждого из этих классов они указали (и доказали оптимальность в случае однопроцессорной системы) следующие алгоритмы.

1. Для класса статических приоритетов они предложили семейство алгоритмов, названное RM (Rate-Monotonic, «алгоритм монотонных коэффициентов»). Основная идея этих алгоритмов состоит в том, что всем работам назначаются неизменные приоритеты, которые вычисляются по некой формуле (коэффициенту) исходя из известных характеристик работ. Например, в [10] такие коэффициенты обратно пропорциональны периоду возникновения работ. Там доказано, что в однопроцессорном случае, когда директивный интервал для каждой работы равен периоду ее возникновения, подобный RM-алгоритм поиска допустимого расписания является точным, а в [23] рассмотрен случай, когда директивный интервал может быть меньше, чем период возникновения, и для такого случая предложен RM-алгоритм, коэффициенты приоритетов работ в котором обратно пропорциональны соответствующим дедлайнам.

2. Для класса динамических приоритетов было предложено семейство алгоритмов, названное EDF (earliest deadline first, «вначале - ближайший срок завершения»). Основная идея этих алгоритмов заключается в том, что самой приоритетной в каждый момент времени среди доступных в этот момент работ считается та работа, директивный срок окончания которой наступит раньше всех остальных. В [14] показано, что в однопроцессорном случае оптимальным с точки зрения минимизации общего времени выполнения работ будет являться то расписание, которое в каждый момент времени выполняет работу, директивный срок завершения выполнения которой наступит ранее остальных.

В данной работе мы будем рассматривать, главным образом, апериодический случай, поэтому многие из предложенных алгоритмов будут основаны на идее EDF. Идеи двух классов алгоритмов, предложенных Лю и Лейландом, оказались очень жизнеспособными. Например, один из RM-алгоритмов был выбран для составления расписаний на выполнение множества независимых автоматических работ, возникающих на Международной Космической Станции. Этот алгоритм был встроен в бортовую вычислительную систему и напрямую поддерживался используемым там компилятором языка Ада.

Несмотря на то, что для однопроцессорной системы RM является оптимальным алгоритмом в случае периодически возникающих задач, а EDF – в апериодическом случае, ни один из них не является оптимальным в случае многопроцессорной системы. Изолированная задача распределения памяти при заданном расписании выполнения работ в однопроцессорной системе реального времени была изучена Сушковым и Логиновой в [17] и [49].

В многопроцессорном случае известно существенно меньше теоретических результатов, чем в однопроцессорном, во многом по той причине, что многие частные случаи задачи составления допустимого многопроцессорного расписания являются NP-трудными. В силу этого, как указано в [7], актуальным в настоящее время подходом, имеющим большое практическое значение, является рассмотрение упрощенных постановок и частных случаев исходной задачи, а также построение, наряду с точными, эвристических алгоритмов сравнительно невысокой вычислительной сложности. Два этих подхода и применяются в данной работе для построения алгоритмов, решающих задачу о многопроцессорном расписании в системе жесткого реального времени в ее различных вариантах.

Наряду с задачей построения допустимого расписания для известной вычислительной системы реального времени, актуальной является также и обратная задача построения (синтеза) системы реального времени некоторой минимально возможной конфигурации, в которой всегда может быть найдено допустимое расписание при заданном наборе работ. Особенно актуальной эта проблема является для бортовых вычислительных комплексов, при проектировании которых обычно стремятся минимизировать необходимые вычислительные ресурсы с целью экономии их массы и потребляемой электроэнергии.

Решение задачи построения вычислительных систем выполняется, как правило, поэтапно. В соответствии с рассматриваемой детализацией аппаратных и программных средств вычислительной системы выделяется несколько уровней проектирования. Как правило, таких уровней три: абстрактный, системный и уровень регистровых передач [25, 26, 27, 28]. На абстрактном уровне рассматриваются ограничения на сроки выполнения прикладной программы и аппаратные ресурсы и

анализируется возможность построения системы с учётом выполнения этих ограничений. На системном уровне определяются характеристики основных структурных компонентов вычислительной системы, например, процессоров и устройств ввода-вывода. На уровне регистровых передач происходит проектирование системы с использованием конкретной элементной базы для дальнейшей непосредственной реализации системы. К задачам регистрового уровня построения вычислительных систем относятся, например, задачи автоматического проектирования микросхем, трассировки плат и размещения микросхем на плате.

К настоящему времени уже разработан ряд промышленных систем, автоматизирующих решение задачи синтеза вычислительных систем на уровне регистровых передач [29, 30, 31]. Однако методов, полностью автоматизирующих решение задачи построения структур вычислительных систем на системном и абстрактном уровне, на данный момент не известно. Указанные причины обуславливают актуальность задачи разработки методов, которые позволили бы автоматизировать и ускорить процесс синтеза структур вычислительных систем. В [18] приведены ограничения на необходимые и достаточные производительности процессоров системы жесткого реального времени в случае периодически возникающих работ и отсутствия ограничения по памяти процессоров, задача синтеза системы в этом случае свелась к задаче целочисленного линейного программирования, в [32], [33], [34] и [35] предложены генетические алгоритмы синтеза рассматриваемых систем.

Целями настоящей работы являются:

- построение математических моделей многопроцессорных систем жесткого реального времени без ограничений по памяти процессоров; разработка и анализ эффективных эвристических алгоритмов решения задачи поиска допустимого расписания с прерываниями и без прерываний в этих системах;

- получение аналитических ограничений на существование допустимого расписания в математических моделях систем реального времени с ограничениями по памяти процессоров и пропускной способности канала ввода-вывода, построение и анализ точных и эври-

стических алгоритмов построения допустимого расписания в таких системах;

- получение необходимых и достаточных условий на параметры систем жесткого реального времени с ограничениями по памяти процессоров для существования в ней допустимого расписания при заданном наборе работ, построение алгоритмов поиска этих параметров (решение задачи синтеза);

- разработка и анализ точных и эвристических алгоритмов решения задачи поиска оптимального расписания в этих системах;

- разработка алгоритмов решения задачи составления расписания путем агрегирования заданий системы и решения полученной таким образом задачи меньшей размерности;

- разработка параллельных алгоритмов решения задачи составления расписания большой размерности с высоким коэффициентом эффективности и линейной функцией изоэффективности;

- получение аналитических ограничений на точность расписания, получаемого эвристическими алгоритмами, и на время выполнения алгоритмов;

- разработка алгоритмов составления допустимого расписания с прерываниями в многопроцессорной системе в случае, когда заданы директивные интервалы, а длительности выполнения работ линейно зависят от количества выделенного им дополнительного ресурса;

- разработка алгоритмов составления многопроцессорных периодических расписаний с прерываниями;

- разработка алгоритмов составления многопроцессорных расписаний при наличии неопределенных факторов;

- разработка алгоритмов организации контроля в системах реального времени.

В работе использованы материалы из публикаций, написанных автором совместно с Д.С. Гузом (глава 1). Д.В. Красовским (глава 2), Е.О. Косоруковым (разд. 3.1) и Б.В. Гречуком (глава 4), которыми также было разработано программное обеспечение, проведены машинные эксперименты и оформлены их результаты в виде таблиц, графиков и диаграмм. Автор выражает благодарность Д.Р. Гончару за большую помощь при подготовке книги к печати.

Глава 1.
Разработка алгоритмов составления многопроцессорных расписаний с прерываниями и синтеза систем реального времени

Настоящая глава посвящена

- построению математических моделей многопроцессорных систем жесткого реального времени без ограничений по памяти процессоров; разработке и анализу эффективных эвристических алгоритмов решения задачи поиска допустимого расписания с прерываниями в этих системах;

- получению аналитических ограничений на существование допустимого расписания в математических моделях систем реального времени с ограничениями по памяти процессоров и пропускной способности канала ввода-вывода, построению и анализу точных и эвристических алгоритмов нахождения допустимого расписания в таких системах;

- получению необходимых и достаточных условий на параметры систем жесткого реального времени с ограничениями по памяти процессоров для существования в ней допустимого расписания при заданном наборе работ, построению алгоритмов поиска этих параметров (решение задачи синтеза).

В разд. 1.1. дается постановка задачи.

В разд. 1.2 рассматривается задача поиска решений в многопроцессорной системе реального времени, в которой разрешены прерывания и переключения работ, переключения работ не требуют дополнительныхъ затрат, связи между процессорами образуют полный граф, а ограничения по памяти отсутствуют. Для этого случая приводится известный точный полиномиальный алгоритм, который, однако, затруднительно применять на практике в системах большой размерности из-за высокой степени полинома его вычислительной сложности. Поэтому для данного случая предлагаются два более быстрых эвристических алгоритма, приводятся результаты их испытаний, проводится их сравнительный анализ и даются рекомендации по практическому применению.

В разд. 1.3 условия задачи из первой главы усложняются введением временных затрат процессоров на прерывания и возобновления работ. Рассматривается три различных модели образования таких затрат, при условии, что во всех этих случаях затраты на прерывания и возобновления малы относительно характерной длины директивных интервалов работ. Для решения задачи в этом случае предлагаются модификации предложенных в разд. 1.2 эвристических алгоритмов, исследуются области предпочтительности применения каждого из предложенных алгоритмов.

В разд. 1.4 рассматривается задача о поиске допустимого расписания в многопроцессорной системе с ограничениями по памяти процессоров, в которой процессоры, помимо скорости выполнения работ, характеризуются скоростью загрузки данных. Рассматривается случай одинаковых директивных интервалов для всех работ. Для однопроцессорного случая строится точный алгоритм, решающий задачу за полиномиальное время. Далее рассматривается многопроцессорный случай этой же задачи как в случае возможности прерывания и переключения работ, так и в случае отсутствия этой возможности. Предлагается эвристический алгоритм, решающий задачу за полиномиальное время, исследуются границы его корректной применимости.

В разд. 1.5 рассматривается задача поиска допустимого расписания в многопроцессорной системе реального времени с ограничениями по объему памяти процессоров в случае неполного графа связей между процессорами, которые работают дискретными синхронизованными по времени тактами. Для решения задачи строится многопродуктовая потоковая сеть специального вида. Приводятся необходимые и достаточные условия существования допустимого расписания в виде условий существования многопродуктового потока в построенной сети, удовлетворяющего определенным ограничениям, описывается, как из полученного потока строится допустимое расписание, указываются наиболее эффективные алгоритмы поиска такого потока.

В разд. 1.6 рассматривается задача синтеза. Приводится система неравенств, задающая допустимую с точки зрения существования допустимого расписания область производительности процессоров в

случае, когда ограничения на объем памяти процессоров отсутствуют, и методы решения возникающих в связи с этим различных оптимизационных задач. Далее рассматривается задача синтеза системы реального времени в случае ограничений по объему памяти процессоров, строится система неравенств, связывающая искомые производительности процессоров и их объемы памяти с параметрами работ, являющаяся необходимым и достаточным условием для существования в рассмотренной системе допустимого расписания с прерываниями, приводятся методы решения возникающих здесь различных оптимизационных задач.

Доказательства приводимых в главе 1 утверждений содержатся в [18, 37, 42, 43, 44, 47, 48].

1.1. Постановка задачи

Сформулируем рассматриваемую в данной главе задачу о допустимом многопроцессорном расписании для системы жесткого реального времени следующим образом.

Рассматривается вычислительная система, состоящая из m процессоров. Каждый процессор j ($j = 1,...,m$) характеризуется скоростью выполнения работ s_j, скоростью загрузки данных l_j и объемом памяти V_j. Имеется n работ, каждая из которых определяется своим директивным сроком начала r_i и директивным сроком окончания d_i, другими словами, директивным интервалом $(r_i, d_i]$, $i = 1,...,n$, а также сложностью p_i, $i = 1,...,n$ и объемом памяти v_i, который необходимо загрузить в процессор для ее выполнения. Работа сложности p_i может быть выполнена на процессоре j за время $\frac{p_i}{s_j}$. Время загрузки данных i-й работы на j-й процессор равно $\frac{v_i}{l_j}$. До начала выполнения работы i в память процессора j, на котором она будет выполнена, должны быть загружены соответствующие этой работе данные. Все параметры задачи полагаются целыми. Загрузка данных может производиться на каждом процессоре одновременно с выполнением какого-либо задания. Память может быть загружена (частично или полностью) некоторыми

данными уже в начальный момент времени. Удаление данных из памяти происходит мгновенно сразу после выполнения соответствующей работы. В фиксированный момент времени каждая работа может выполняться не более чем одним процессором, и каждый процессор может выполнять не более одной работы. При выполнении работ допускаются прерывания и переключения с одного процессора на другой. Предполагается, что прерывание выполнения работы на процессоре *j* с целью ее переключения на одном из последующих тактов на другой процессор или для возобновления на этом же процессоре требует временных затрат системы, равных λ_j. Граф связей между процессорами (отражающий возможность переключения работы с одного процессора на другой) произвольный.

Задача состоит в том, чтобы определить, существует ли расписание, позволяющее выполнить все работы в их директивные интервалы на имеющихся процессорах и, в случае положительного ответа, указать такое расписание. Будем называть в дальнейшем такое расписание *допустимым расписанием*, а саму задачу – поиском *допустимого расписания*.

Мы привели самую общую, наиболее сложную постановку задачи составления расписания в системе жесткого реального времени, однако заметим, что даже существенно более простые ее частные случаи являются NP-трудными задачами [1].

Лемма 1. Вариант рассматриваемой задачи о многопроцессорном расписании, в котором прерывания задач требуют процессорного времени, ограничений на объем памяти нет, а связи между процессорами образуют полный граф, является NP-трудной задачей.

Таким образом, задача о поиске допустимого расписания в своей общей постановке сложна для построения эффективных алгоритмов ее решения, поэтому в данной работе рассматриваются различные варианты упрощения условий исходной задачи, а также частные случаи, которые, тем не менее, все еще имеют обширную практическую интерпретацию. Другим предлагаемым подходом является применение эвристических алгоритмов. Далее рассмотрим следующий относи-

тельно простой вариант сформулированной выше задачи и предложим ряд эффективных алгоритмов для его решения.

1.2. Связи – полный граф. Отсутствие ограничений по памяти. Переключения без затрат

Сформулируем рассматриваемую в данном разделе задачу о допустимом многопроцессорном расписании следующим образом. Имеется t типов процессоров, каждый из которых характеризуется скоростью s_i ($i = 1...t$), всего имеется m_i ($i = 1...t$) процессоров каждого типа. Связи между процессорами образуют полный граф. Имеется n работ, каждая из которых определяется своим директивным сроком начала r_j и директивным сроком окончания d_j, другими словами директивным интервалом $(r_j, d_j]$, $j = 1...n$, а также сложностью p_j, $j = 1...n$. Работа сложности p_j может быть выполнена на процессоре типа i за время $\frac{p_j}{s_i}$. При выполнении работ допускаются прерывания и переключения с одного процессора на другой. Прерывания и переключения работ не требуют временных затрат системы.

1.2.1. Точный алгоритм

Начнем с построения алгоритма, находящего точное решение задачи. В качестве такого алгоритма была построена комбинация быстрых в своем классе алгоритмов, предложенных в [11] и [12], реализованная следующим образом.

Пусть $\tau_0 < \tau_1 < ... < \tau_k$ – все различные значения r_j и d_j, $j = 1,\ldots,n$, $I_i = (\tau_{i-1}; \tau_i]$, $i = 1,\ldots,k$. По условиям задачи, работа j доступна для выполнения (*выполнима*) в момент времени t_0, если $r_j \le t_0$ и $d_j \ge t_0$. Заметим, что внутри каждого интервала I_i набор выполнимых работ остается постоянным. Поэтому основная идея предлагаемого подхода состоит в том, что сначала определяется, какое количество (то есть, какую часть от величины p) каждой работы следует выполнить на каждом интервале I_i. Затем рассматривается отдельно каждый интервал

I_i как сужение задачи о многопроцессорном расписании до случая, когда у всех работ один и тот же директивный интервал.

Методика распределения частей работ по интервалам приведена в [11]. В этой работе доказывается, что такая задача может быть сведена к задаче поиска максимального потока в сети, построенной специальным образом. Построим такую сеть (*N*,*E*).

Сеть представляет из себя трехдольный граф с дополнительными источником *O* и стоком *O´*. Первый уровень графа состоит из вершин, каждая из которых взаимно однозначно соответствует какой-либо работе (будем называть их «вершины работ»), второй уровень состоит из вершин, которые соответствуют всем возможным комбинациям различных процессоров и интервалов (будем называть их «вершины процессора-интервала»), третий уровень состоит из вершин, соответствующих каждому интервалу (назовем их «вершины-интервалы»). Источник соединен со всеми вершинами работ, причем дуга, ведущая к вершине *j*-ой работы, имеет пропускную способность, равную p_j, $j = 1...n$. Вершина работы соединяется со всеми вершинами процессора-интервала, которые отвечают за те интервалы, на которых эта работа выполнима. Пропускная способность всех дуг, которые ведут в вершину процессора-интервала, соответствующую процессору типа *r* и интервалу длины *T*, равна $(s_r - s_{r+1})T$. Считаем, что $s_{t+1} = 0$. Каждая вершина процессора-интервала (*r*,*i*), соответствующая процессору типа *r* и интервалу длины T_i соединена с вершиной интервала *i* дугой емкости $\left(\sum_{n=1}^{r} m_n\right)(s_r - s_{r+1})T_i$. В заключение, все вершины интервалов соединены со стоком *O´* дугами бесконечно большой пропускной способности. На рис.1 приведено строение подобной сети.

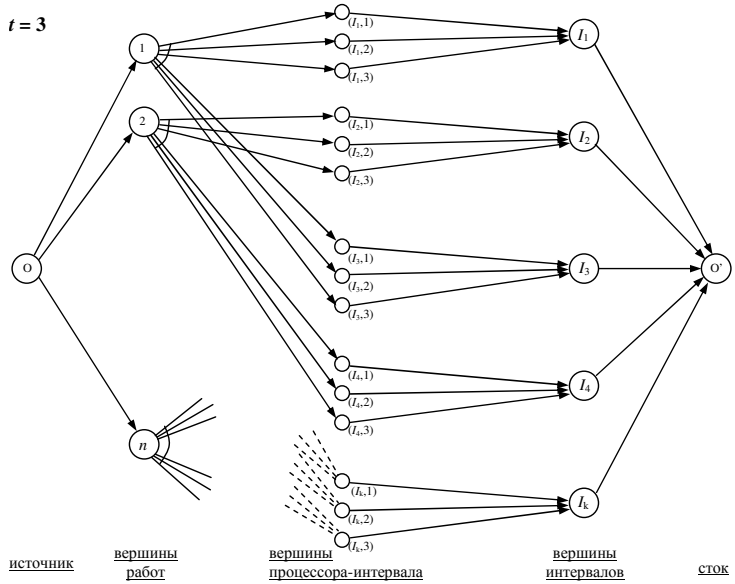


Рис 1. Сеть (N,E).

Найдем число вершин в построенной таким образом сети: вершины-работы (n вершин), вершины процессоры-интервалы ($(2n-1)t$ вершин), вершины-интервалы ($2n$ - 1 вершин). Всего получаем $N = O(tn)$ вершин. Теперь оценим сверху возможное в построенной сети (N, E) количество дуг: дуги, соединяющие источник и вершины работы (n дуг), дуги, соединяющие вершины работы и вершины процессоры-интервалы (возможно соединение каждой вершины работы с каждой вершиной процессора-интервала) (не более $(2n-1)nt$ дуг), дуги, соединяющие вершины процессора-интервала с вершинами интервала ($(2n-1)t$ дуг). Отсюда получаем $|E| = O(tn^2)$ дуг.

Для поиска максимального потока в построенной сети воспользуемся алгоритмом «поднять-и-в-начало», описанным в [13]. Этот алгоритм использует идею *предпотока*, предложенную Карзановым, и представляет собой развитие классического алгоритма Гольдберга «проталкивания предпотока». Эффективность метода «поднять-и-в-начало» - $O(|N|^3)$ [13]. Так как ранее было замечено, что $N = O(tn)$, то эффективность этой фазы алгоритма равна $O(t^3n^3)$. В [11] доказывает-

ся, что допустимое расписание, являющееся точным решением классической задачи о многопроцессорном расписании, существует тогда и только тогда, когда максимальный поток в построенной таким образом сети (*N*,*E*) равен $\sum_{j=1}^{n} p_j$, то есть пропускной способности разреза $(s, N \setminus s)$. Если полученный максимальный поток удовлетворяет этому свойству, возникает задача определения допустимого расписания. По виду полученного максимального потока можно определить, какая часть от каждой работы должна быть выполнена на каждом из интервалов I_i. Для каждой работы достаточно проследить, какие части максимального потока, направленного по дуге, соединяющей исток с вершиной этой работы, попадают в результате на связанные с этой вершиной вершины интервалов (см. рис. 2).

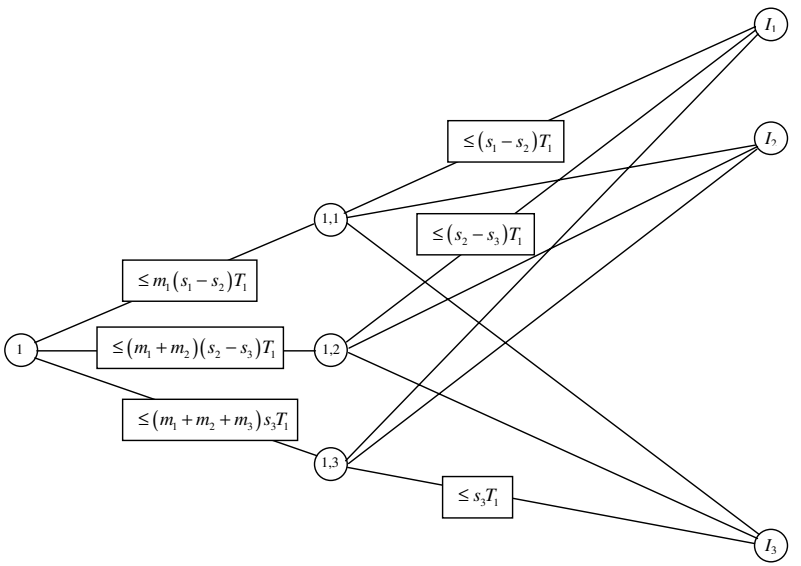


Рис 2. Распределение частей работ по интервалам.

Определив распределение частей работ по всем интервалам, получаем серию локальных задач (для каждого интервала), являющихся еще одним более простым, чем рассматривается в это главе, частным

случаем задачи о многопроцессорном расписании, когда директивные интервалы всех работ равны длине этого интервала. Результат [11] гарантирует, что каждая такая локальная задача имеет своим решением допустимое расписание. Для нахождения этого расписания в работе использовался один из наиболее эффективных алгоритмов для этого частного случая, описанный и обоснованный в [12]. Там же доказывается, что эффективность алгоритма поиска точного решения задачи о многопроцессорном расписании в случае одинаковых директивных интервалов работ равна $O(n)$. В составленном допустимом расписании этот алгоритм допускает не более $2(m\text{-}1)$ прерываний, где m – общее число процессоров, т.е. $m = \sum_{i=1}^{t} m_i$.

Во второй фазе предлагаемого точного решения, в случае выполнения условия существования в сети N максимального потока величины $\sum_{j=1}^{n} p_j$ ([11]), алгоритм из [12] должен быть применен на каждом интервале, то есть $2n\text{-}1$ раз. Отсюда эффективность второй фазы алгоритма равна $O(n(2n\text{-}1)) = O(n^2)$. Следовательно, полное время выполнения предлагаемого алгоритма поиска точного решения равно $O(t^3n^3+n^2) = O(t^3n^3)$.

Оценим количество прерываний, которое допускает предложенный метод. В [12] дается оценка для каждого интервала, равная $2(m\text{-}1)$, значит, всего мы получаем $2(2n\text{-}1)(m\text{-}1)$ возможных прерывания внутри интервалов. Помимо прерываний внутри интервалов, возможны прерывания на границах интервалов, причем в предельном случае у каждой работы (которых n штук). Всего границ интервалов в построенной системе $2(n\text{-}1)$, поэтому получаем дополнительно еще $2n(n\text{-}1)$ прерываний. Таким образом, предложенный метод поиска точного решения задачи допускает не более чем $2(2n\text{-}1)(m\text{-}1)+2n(n\text{-}1)= = 2(n^2+2mn-3n-m+1)$ прерываний.

Итак, предложенный алгоритм находит точное решение сформулированного в этой главе варианта задачи о многопроцессорном расписании (полный граф связей между процессорами, ограничения по памяти отсутствуют, прерывания работ не требуют временных затрат)

за время $O(t^3n^3)$, допуская при этом не более $2(n^2+2mn-3n-m+1)$ прерываний.

Несмотря на то, что предложенный алгоритм находит точное решение за полиномиальное время ($O(t^3n^3)$), тем не менее это время является слишком большим для практического применения данного метода в задачах достаточно большой размерности, имеющих практический смысл. Во время испытаний на компьютере с процессором Pentium III 866 MHz этот алгоритм показывал времена, приведенные в табл. 1. Из построенной таблицы хорошо заметно, что уже на задаче расчета расписания на 64 процессорах (вполне реальный случай) алгоритм показывает недопустимо долгое время работы. Одним из решений возникшей проблемы может быть разработка существенно более быстрого эвристического алгоритма, который находил бы правильное решение задачи в достаточно широком спектре случаев, и описание этого спектра, то есть границ применимости разработанного эвристического алгоритма.

Таблица 1. Среднее время работы точного алгоритма в зависимости от размерности задачи.

Размерность задачи		*Среднее время работы*
Число процессоров	*Число работ*	
4	10	1.23 с
8	25	14.32 с
16	50	1090.87 с
64	500	11 ч.

Далее предлагается описание и анализ двух эвристических алгоритмов, находящих решение задачи о построении допустимого расписания в варианте, сформулированном в начале данного раздела. Оба эти алгоритма были разработаны как варианты обобщения алгоритма Коффмана [14], предназначенного для решения задачи на одном процессоре, на несколько процессоров с различными производительностями. Назовем эти два алгоритма *Эвристика 1* и *Эвристика 2* и рассмотрим их работу подробнее.

1.2.2. Эвристические алгоритмы

1.2.2.1. Эвристика 1

Описание алгоритма:

Процедура 1. Пусть $\tau_0 < \tau_1 < ... < \tau_k$ – все различные значения r_j и d_j, $j = 1,\ldots,n$. Будем двигаться по временной оси от τ_0 ($\min_j r_j$) к τ_k ($\max_j d_j$). Обозначим множество доступных в момент времени t работ как $D(t)$, множество выполняемых в этот момент времени работ как $C(t)$. Введем вектор-функцию $\mathbf{z}(t)$, каждый элемент которой $z_j(t)$ равен объему соответствующей работы, который осталось выполнить в момент времени t. Определим функции $\delta_j(t) = \begin{cases} 1, j \in C(t) \\ 0, j \notin C(t) \end{cases}$ и $S_j(t)$, значение которой равно скорости процессора, на который в момент времени t назначена работа j (если в этот момент времени работа j никуда не назначена, полагаем $S_j(t) = 0$). В начале работы алгоритма U состоит из единственного элемента $\min_j r_j$, K – пустое множество, $\mathbf{z}(t) = \|p_j\|$.

Процедура 2. Пусть в момент времени t мы достигли какого-то из r_j, либо какая-то из работ оказалась полностью выполненной на каком-либо процессоре. Если оказалось, что в этот момент времени множество $D(t)\backslash C(t)$ пусто, то мы просто двинемся дальше по времени. Если же оказалось, что в этот момент времени множество $D(t)\backslash C(t)$ не пусто, то каждая работа $j \in D(t)$ будет иметь приоритет, определяемый своим директивным сроком окончания, равный $-d_j$. Если в текущий момент времени имеются свободные процессоры, то на них будут назначаться работы из $D(t)\backslash C(t)$ в порядке убывания приоритета и скорости процессора. Если после этого множество $D(t)\backslash C(t)$ не опустело, но

$$\forall j \in D(t) \setminus C(t) : d_j \geq \max_{i \in C(t)} d_i, \qquad (1)$$

ничего не меняется, и мы движемся дальше по времени. Если же условие (1) не выполнено, то есть для некоторой работы $j \in D(t) \setminus C(t)$ директивный срок ее окончания меньше, чем срок какой-либо из уже назначенных работ, то работа *k*, такая, что $d_k = \max_{i \in C(t)} d_i$, прерывается, и на процессор, выполнявший работу *k*, назначается работа *j*. Эта операция повторяется до тех пор, пока не станет справедливым (1), что будет означать движение дальше по времени до его следующего обрабатываемому алгоритмом момента. Пусть переход по времени совершается на величину $\Delta t = t_{\mathrm{N}} - t$ (t_N – следующий обрабатываемый алгоритмом момент времени). Перед переходом на следующий момент времени t_N (соответственно, на Процедуры 2 или 3 алгоритма в зависимости от того, является ли это время t_N временем окончания выполнения какой-либо из работ или r_j какой-то из работ, либо это время окончания какого-либо директивного интервала d_j), мы (1) фиксируем текущее состояние расписания (*U*,*K*) путем добавления текущего *t* в вектор *U* и очередного столбца, описывающего полученное в этот момент распределение работ по процессорам, в матрицу *K*; (2) получаем новый вектор $\mathbf{z}(t_{\mathrm{N}})$ путем вычисления каждого его компонента следующим образом:

$$\mathbf{z}(t_{\mathrm{N}}) = \left\| z_j(t_N) \right\| = \left\| z_j(t) - \delta_j(t) \cdot S_j(t) \cdot \Delta t \right\|. \qquad (2)$$

Процедура 3. Пусть в момент времени *t* мы достигли какого-то из d_j. Мы производим проверку, была ли работа *j* полностью выполнена к этому моменту, т.е. выполнено ли для текущего *t* условие $z_j(t) = 0$. Если да, то мы движемся дальше по времени до его следующего обрабатываемому алгоритмом момента, если же нет, то делается вывод, что выполнимого расписания при заданных условиях не существует.

Оценим время выполнения (вычислительную сложность) алгоритма Эвристика 1. Всего имеется *n* работ и *n* директивных сроков начала этих работ, поэтому Процедура 2 этого алгоритма стартует 2*n* раз. Процедура 2 потребует порядка *m* операций сравнения директивных сроков окончания доступных не назначенных работ и уже назна-

ченных на каждый из m процессоров работ, а также порядка m назначений работ на процессоры. Получаем, что Процедура 2 требует $O(mn)$ операций.

Во время выполнения Эвристики 1 Процедура 3 отрабатывает n раз, по числу директивных сроков окончания работ. На этой стадии работы алгоритма просто проверяется ситуация с одной из работ, что составляет одну операцию. Таким образом, всего Процедура 3 требует $O(n)$ операций. Итак, время выполнения Эвристики 1 равно $O(mn + n) = O(mn)$.

Оценим количество прерываний в допустимом расписании, получаемом в результате работы Эвристики 1 в случае, если такое расписание удалось построить. Исходя из определения работы этого алгоритма, можно заметить, что прерывание какой-либо работы на какомлибо процессоре возможно только в момент наступления начала директивного интервала некоторой другой работы (причем прерываемая работа уже должна быть ранее назначена и выполняться), то есть в момент времени r_j, $r_j \neq \min_j r_j$. Так как таких моментов всего n-1, делаем вывод, что Эвристика 1 допускает не более, чем (n-1) прерывание в составленном ею допустимом расписании.

Лемма 2. Сложность предложенного эвристического алгоритма поиска решения задачи о допустимом расписании Эвристика 1 равна $O(nm)$. В составленном допустимом расписании этот алгоритм допускает не более $n-1$ прерываний.

На рис. 3 приведено допустимое расписание, полученное при помощи Эвристики 1. Условия задачи показаны в нижней части рис.3.

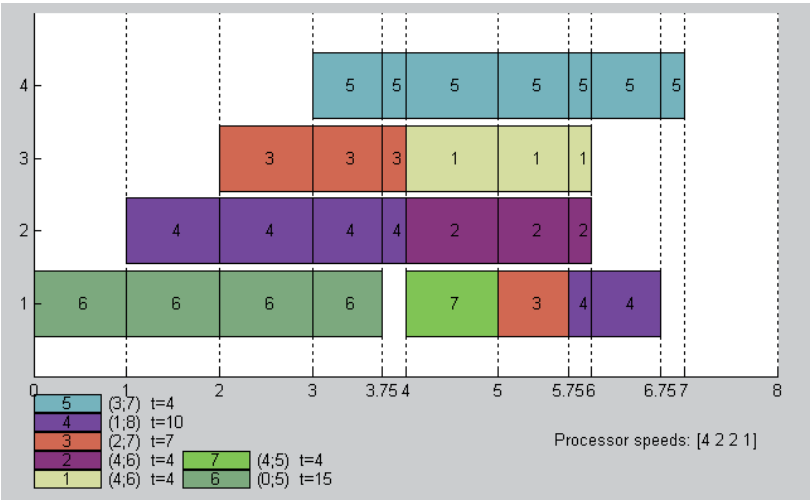


Рис 3. Пример работы алгоритма Эвристика 1.

1.2.2.2. Эвристика 2

Алгоритм Эвристика 2 отличается от алгоритма Эвристика 1 только процедурой 2. Если в Эвристике 1 во время выполнения Процедуры 2 с процессоров снимались только самые низкоприоритетные работы и только в случае возникновения такой необходимости, то в Эвристике 2 мы в каждый обрабатываемый алгоритмом момент времени полностью переназначаем все работы на процессоры в соответствии с их приоритетами и скоростями процессоров. Дадим более формальное описание алгоритма Эвристика 2:

Описание алгоритма:

Процедура 1. Пусть $\tau_0 < \tau_1 < ... < \tau_k$ - все различные значения r_j и d_j, $j = 1,\ldots,n,$. Будем двигаться по временной оси от τ_0 ($\min_j r_j$) к τ_k ($\max_j d_j$). Обозначим множество доступных в момент времени t работ как $D(t)$, множество выполняемых в этот момент времени работ как $C(t)$. Введем вектор-функцию $\mathbf{z}(t)$, каждый элемент которой $z_j(t)$ равен объему соответствующей работы, который осталось выполнить в момент времени t.

Определим функции $\delta_j(t) = \begin{cases} 1, j \in C(t) \\ 0, j \notin C(t) \end{cases}$ и $S_j(t)$, равную скорости процессора, на который в момент времени t назначена работа j (если в этот момент времени работа j никуда не назначена, полагаем $S_j(t) = 0$). В начале работы алгоритма U состоит из единственного элемента $\min_j r_j$, K – пустое множество, $\mathbf{z}(t) = \left\| p_j \right\|$.

Процедура 2. Пусть в момент времени t мы достигли какого-то из r_j, либо какая-то из работ оказалась полностью выполненной на каком-либо процессоре. Приостанавливаем выполнение всех активных работ. После этого $C(t)$ пусто, а $P(t)=\{1,..,m\}$. До тех пор, пока либо $P(t)$, либо $D(t)$ не станет пустым множеством, выполняем следующую последовательность действий: (1)назначаем j-ю работу, такую, что $d_j = \min_{i \in D(t)} d_i$, на процессор k, удовлетворяющий условию: $s_k = \max_{i \in P(t)} s_i$.; $P(t) \leftarrow P(t) \setminus \{k\}$; $D(t) \leftarrow D(t) \setminus \{j\}$; $C(t) \leftarrow C(t) \bigcup \{j\}$.

Как только либо $P(t)$, либо $D(t)$ станет пустым множеством, это означает, что, соответственно, либо больше нет свободных процессоров, либо нет доступных работ, и нужно переходить на следующий временной шаг. Пусть переход по времени совершается на величину $\Delta t = t_N - t$ (t_N – следующий обрабатываемый алгоритмом момент времени). Перед переходом на следующий обрабатываемый алгоритмом момент времени t_N (соответственно, на Процедуры 2 или 3 алгоритма Эвристика 2 в зависимости от того, является ли этот момент времени t_N временем окончания выполнения какой-либо из работ или начала директивного интервала r_j некоторой работы, либо это время окончания какого-либо директивного интервала d_j), мы: (1) фиксируем текущее состояние расписания (U, K) путем добавления текущего рассматриваемого момента времени t в вектор U и очередного столбца, описывающего полученное в этот момент распределение работ по процессорам, в матрицу K; (2) получаем новый вектор $\mathbf{z}(t_N)$ путем вычисления каждого его компонента следующим образом:

$$\mathbf{z}(t_N) = \left\| z_j(t_N) \right\| = \left\| z_j(t) - \delta_j(t) \cdot S_j(t) \cdot \Delta t \right\|. \qquad (3)$$

Процедура 3. Пусть в момент времени t мы достигли какого-то из d_j. Производится проверка, была ли работа j полностью выполнена к этому моменту, т.е. выполнено ли для текущего t условие $z_j(t) = 0$. Если да, то мы двигаемся дальше по времени, если же нет, то делается вывод, что допустимого расписания при заданных условиях не существует.

Лемма 3. **С**ложность предложенного эвристического алгоритма поиска решения задачи о допустимом расписании Эвристика 2 равна $O\left(n^2 \log n\right)$. В составленном допустимом расписании этот алгоритм допускает не более $2m(n\text{-}1)$ прерываний.

Таким образом, алгоритм Эвристика 2 допускает существенно больше прерываний, чем алгоритм Эвристика 1. Для более наглядного сравнения двух предложенных алгоритмов приведем здесь изображение допустимого расписания, полученного при помощи алгоритма Эвристика 2 для тех же условий задачи, которые использовались для иллюстрации работы алгоритма Эвристика 1 (рис. 4).

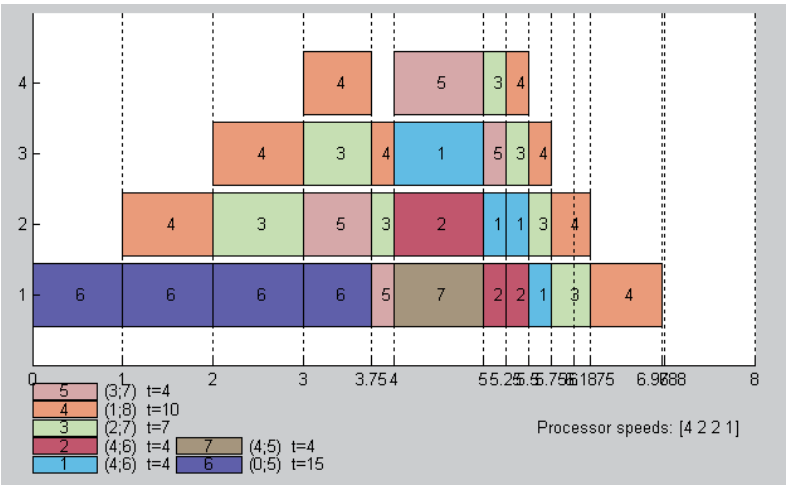


Рис 4. Пример работы алгоритма Эвристика 2.

1.2.3. Сравнительный анализ алгоритмов Эвристика 1 и Эвристика 2

Оба эвристических алгоритма были реализованы в виде независимых модулей на языке MatLab. Каждый из этих модулей принимает на входе набор векторов, задающих условия задачи о многопроцессорном расписании в рассматриваемом в данной главе варианте, а на выходе выдает сообщение о том, что алгоритм не нашел допустимого расписания, либо пару (U, K), описывающую получившееся многопроцессорное расписание.

Ранее мы получили, что вычислительные сложности алгоритмов Эвристика 1 и Эвристика 2 соответственно равны $O(nm)$ и $O(n^2\log n)$. Насколько это большой выигрыш во времени по сравнению с алгоритмом, находящим точное решение, (сложность $O(t^3n^3)$) хорошо заметно по приведенной ниже сравнительной таблице (табл. 2), составленной путем усреднения времени выполнения алгоритмов по большому количеству численных экспериментов:

Таблица 2. Среднее время работы алгоритмов Эвристика 1, Эвристика 2 и точного алгоритма в зависимости от размерности задачи.

Размерность задачи		Среднее время работы		
Число процессоров	Число работ	Эвристика 1	Эвристика 2	Точное решение
4	10	0.03 с	0.06 с	1.23 с
8	25	0.11 с	0.32 с	14.32 с
16	50	0.28 с	1.36 с	1090.87 с
16	100	1.23 с	4.86 с	6311.1 с
64	500	3.45 с	14.6 с	11 ч.

Исследуем теперь границы применимости обоих предложенных эвристических алгоритмов.

Лемма 4. Возможен только один вариант некорректной работы предложенных эвристических алгоритмов: при существовании допустимого расписания алгоритм утверждает, что его нет.

Таким образом, проблема поиска границ применимости эвристических алгоритмов в данном случае фактически сводится к нахождению области, в которой эвристический алгоритм дает отрицательный результат, а точный алгоритм находит допустимое расписание. Для выявления границ применимости эвристик были проведены серии численных экспериментов. Каждый эксперимент заключался в запуске при одних и тех же условиях последовательно Эвристики 1, Эвристики 2 и алгоритма поиска точного решения.

Размерность задачи (число процессоров и число заданий) оставалась неизменной для всех экспериментов, а остальные параметры подвергались рандомизации для того, чтобы получить достаточно репрезентативную выборку для обоснованных суждений о границах применимости предложенных эвристических алгоритмов. Также для возможности адекватного сравнения результатов всех экспериментов, нужно, чтобы они проходили в одном и том же временном масштабе. Для этого заранее определяются и остаются неизменными для всех экспериментов $\min_i r_i$ и $\max_i d_i$. Размерность задачи была выбрана таким образом, чтобы с одной стороны быть достаточно большой для представления репрезентативной статистики, с другой – позволять проводить каждый численный эксперимент за обозримое время. В качестве размерности, удовлетворяющей обоим критериям, было выбрано 16 процессоров и 50 задач. Такая размерность позволяла ограничить время самого ресурсоемкого эксперимента (то есть такого, где и Эвристика 1, и Эвристика 2 дали отрицательный результат, и пришлось применять точное решение) получасом расчетов на применявшейся системе. Временной масштаб задачи для всех экспериментов задавался следующим образом:

$$\left| \max_i d_i - \min_i r_i \right| < 50, \text{ а именно } \min_i r_i = 0 \text{ и } \max_i d_i = 50.$$

Каждая серия экспериментов была реализована в виде отдельного модуля MatLab. Целью каждой серии было проследить, как сказываются изменения какого-либо из приведенных в табл. 3 параметров на корректность работы алгоритмов Эвристика 1 и Эвристика 2. Для этого выбирались один или два связанных параметра, и отслеживались

результаты при их последовательном изменении с некоторым малым шагом (остальные параметры фиксировались). При каждом наборе значений параметров проводилось по 10 экспериментов для создания репрезентативной выборки. Шаг изменения выбирался достаточно малым для того, чтобы можно было максимально точно указать границы, внутри которых Эвристики 1 и 2 дают отрицательный результат, тогда как точное решение свидетельствует об обратном. Тем не менее, "грубо" найденная граница всегда пересекалась той же серией экспериментов еще раз с еще более мелким шагом для ее уточнения. Результатом эксперимента считалась тройка, встретившаяся среди 10 испытаний не менее 7 раз. Если какой-либо из экспериментов не давал такую тройку, то в окрестности набора параметров этого эксперимента проводилась дополнительная серия экспериментов с еще более мелким шагом. Ниже в качестве примеров приведены результаты нескольких наиболее характерных из проведенных экспериментов.

Общее количество проведенных численных экспериментов составило около 25'000. В приведенной ниже табл. 3 показано, какой процент от общего числа испытаний составляют эксперименты, в которых эвристические алгоритмы Эвристика 1 и Эвристика 2 отработали некорректно:

Таблица 3. Статистика некорректной работы алгоритмов Эвристика 1 и Эвристика 2, собранная на основании проведенных численных экспериментов.

Размерность задачи		% некорректной работы	
Число процессоров	Число работ	Эвристика 1	Эвристика 2
4	10	18%	2%
8	25	19%	3%
16	50	22%	2%
16	100	19%	1%
64	500	17%	2%

Все проведенные серии численных экспериментов показали (и это также заметно по вышеприведенным характерным примерам), что область некорректного поведения алгоритмов Эвристика 1 и Эвристика 2 компактна. Второе очевидное наблюдение – алгоритм Эвристика 2 работает некорректно на существенно меньшем множестве параметров, чем алгоритм Эвристика 1. Более того, изменения параметров, касающиеся отношений между максимально и минимально возможными значениями параметров, никак не сказывается на корректности Эвристики 2, чего не скажешь об Эвристике 1. Поэтому, несмотря на то, что Эвристика 2 требует несколько больше вычислительных ресурсов, чем Эвристика 1 ($O(n^2 \log n)$ против $O(mn)$), а также в генерирует расписание с большим количество прерываний $(2(n-1)m$ против $(n-1))$, Эвристика 2 представляется более предпочтительным алгоритмом для решения сформулированного в данной главе варианта задачи о поиске допустимого расписания (связи между процессорами – полный граф, ограничения по памяти отсутствуют, временные затраты на прерывания и переключения работ пренебрежимо малы).

И еще одно замечание по поводу применения эвристических алгоритмов. Задача формулируется таким образом, что некорректное поведение такого алгоритма может заключаться только в том, что он выдаст заключение об отсутствии допустимого расписания там, где оно на самом деле есть. При решении практических задач отсутствие допустимого расписания при заданных значениях параметров обычно означает только то, что для получения допустимого расписания следует несколько изменить значение параметров задачи. При таком изменении параметров, если мы до этого пребывали в области некорректной работы Эвристики 2, мы с большой вероятностью из нее выйдем и получим допустимое расписание, так как эта область, как показали эксперименты, сравнительно невелика. Альтернативным подходом (если есть необходимость в абсолютной точности ответа на поставленную задачу) может быть следующая процедура: эффективно находить допустимые расписания при помощи алгоритмов Эвристика 1 и Эвристика 2, лишь иногда (когда оба эвристических алгоритма дали отрицательный ответ, но есть подозрение, что допустимое распи-

сание все-таки существует при заданном наборе параметров) прибегая к "точному" алгоритму.

1.3. Переключения с затратами. Связи – полный граф. Отсутствие ограничений по памяти

При рассмотрении и анализе алгоритмов Эвристика 1 и Эвристика 2, проведенном выше, считалось, что прерывания работ не приводят к издержкам процессорного времени. В то же время понятно, что в реальности на любую проводимую операцию многопроцессорная система затрачивает определенное количество времени. Заметим, что при некоторых условиях (когда временные затраты процессоров на прерывания пренебрежимо малы по сравнению с длиной характерного временного интервала I_i, введенного в разд. 1.2.1) такой подход имеет право на существование. Для более точного и реалистичного построения допустимых расписаний желательно (а в случае, когда временные затраты процессоров на прерывания сравнимы с характерной величиной I_i, просто необходимо) учитывать временные затраты процессоров на прерывания. Как уже было отмечено ранее, точное решение задачи поиска допустимого расписания в случае временных затрат на прерывания работ является NP-трудной задачей (лемма 1), и в этом разделе рассматриваться не будет. Будем считать, что время, требуемое на прерывания, расходуется на то, чтобы выгрузить с процессора информацию по незавершенной работе, необходимую для возобновления и успешного завершения ее обработки, в некоторую внешнюю память, а также на загрузку этой информации обратно на процессор при возобновлении обработки прерванной работы.

Рассмотрим поведение предложенных в предыдущей главе алгоритмов Эвристика 1 и Эвристика 2 (точнее, их модификаций для рассматриваемого в данном разделе случая задачи о поиске допустимого расписания, использующих аналогичные принципы) в случае временных затрат на прерывания работ ([38], [39]). Можно сразу сделать предположение, что из-за существенно большего количества прерываний, которое генерирует алгоритм Эвристика 2 по сравнению с алгоритмом Эвристика 1 ($2(n\text{-}1)m$ против $(n\text{-}1)$), в этом случае не имеет

места однозначная предпочтительность применения Эвристики 2 перед Эвристикой 1, вывод о которой был сделан в предыдущем параграфе. Поэтому становится актуальным вопрос, начиная с какого соотношения между I_i и некоторой величиной, характеризующей среднее время, затрачиваемое на прерывание, модифицированный для рассматриваемого случая алгоритм Эвристика 1 становится более предпочтительным, чем модифицированный алгоритм Эвристика 2, то есть корректно генерирующим допустимое расписание с большей вероятностью. Разберем несколько вариантов механизма образования дополнительных затрат.

Вариант 1. Временные затраты любого процессора на приостановку и возобновление любой из работ равны некоторым постоянным величинам, соответственно $\Omega_п > 0$ и $\Omega_в > 0$. Интерпретацией такой модели может быть ситуация, когда время тратится на сохранение и восстановление данных о прерванной работе откуда-нибудь из внешней памяти и на это расходуются вычислительные ресурсы не процессоров из условий задачи о допустимом расписании, а некоторого отдельного устройства ввода/вывода.

Вариант 2. Временные затраты любого процессора на приостановку и возобновление выполнения любой из работ зависят от вычислительной мощности этого процессора (в первом приближении обратно пропорционально), т.е. для процессора *i*-го типа полагаем их соответственно равными $\frac{\omega_п}{s_i}$ и $\frac{\omega_в}{s_i}$, где $\omega_п$ и $\omega_в$ - некоторые положительные коэффициенты, характеризующие соответственно процессы прерывания и возобновления. В этом случае отдельное устройство ввода/вывода отсутствует, и на сбрасывание и возобновление своих состояний для продолжения прерванных работ свои вычислительные ресурсы тратят сами процессоры, поэтому, чем быстрее процессор, тем быстрее он проделает необходимые операции по выгрузке/загрузке.

Вариант 3. Фактически, комбинация предложенных выше Вариантов 1 и 2. Временные затраты на прерывания работ имеют как постоянную, так и переменную, зависящую от процессора, составляю-

щие. Это значит, что для прерывания работы на процессоре *i*-го типа требуется $\frac{\omega_n}{s_i} + \Omega_n$ времени, а для возобновления – $(\frac{\omega_в}{s_i} + \Omega_в)$. Смысл коэффициентов $\Omega_n, \Omega_в, \omega_n$ и $\omega_в$ при этом сохраняется такой же, как и в Вариантах 1 и 2, а интерпретировать это можно более общим случаем, когда многопроцессорная система для прерывания и возобновления работ использует как независимое от процессоров устройство ввода/вывода, так и вычислительные ресурсы самих процессоров.

Для упрощения анализа будем считать, что во всех трех вариантах $\Omega_n = \Omega_в = \Omega$ и $\omega_n = \omega_в = \omega$, то есть считать процессы ввода и вывода идентичными с точки зрения вычислительных затрат. Обозначим временные затраты *i*-го процессора на прерывания и возобновления работ как T_i (конкретное значение T_i будет зависеть от выбранного варианта модели образования временных затрат).

Заметим, что совмещение алгоритмов Эвристика 1 и Эвристика 2 с вышеизложенными моделями учета дополнительных временных затрат на прерывания и возобновления работ применимы на практике и дают адекватные результаты только в случае, когда эти затраты все еще малы (хотя и не пренебрежимо) по сравнению с характерной длиной временного интервала I_i. Рассмотрим модификации реализации алгоритмов Эвристика 1 и Эвристика 2 при всех вышеуказанных вариантах реализации учета затрат процессорного времени на прерывания. Для этого потребуется лишь дополнить эти алгоритмы несколько более сложными процедурами прерывания и возобновления работ, а также введением некоторых дополнительных структур данных.

Прерывание работы в случае ненулевых затрат на прерывание выполнения работ приводит к тому, что процессор, на который была назначена прерываемая работа некоторое время до прерывания был занят не выполнением работы, а процессом прерывания. Это означает, что *k*-ю компоненту вектора $\mathbf{z}(t)$ нужно увеличить на величину ΔZ_k, равную количеству работы, которую процессор, на котором ее прервали, выполнил бы за время, необходимое для прерывания работы. Эта величина зависит от выбранного варианта модели образования дополнительных временных затрат:

Вариант 1: $\Delta Z_k = \Omega \cdot S_k(J_{j1})$. Вариант 2: $\Delta Z_k = \dfrac{w}{S_k(J_{j1})} \cdot S_k(J_{j1}) = w$.

Вариант 3: $\Delta Z_k = \left(\dfrac{w}{S_k(J_{j1})} + \Omega \right) \cdot S_k(J_{j1}) = w + \Omega \cdot S_k(J_{j1})$.

Аналогичным образом обрабатывается назначение какой-то из прерванных работ на *i*-й процессор (при возобновлении ранее прерванной работы на *i*-м процессоре). Пусть возобновляется работа *j*. Это означает, что *k*-ю компоненту вектора **z**(*t*) нужно увеличить на величину ΔZ_k, равную количеству работы, которую процессор, на котором ее возобновляют, выполнил бы за время, необходимое для возобновления работы. Эта величина зависит от выбранного варианта модели образования дополнительных временных затрат на прерывания работ:

Вариант 1: $\Delta Z_k = \Omega \cdot S_k(t)$. Вариант 2: $\Delta Z_k = \dfrac{w}{S_k(t)} \cdot S_k(t) = w$.

Вариант 3: $\Delta Z_k = \left(\dfrac{w}{S_k(t)} + \Omega \right) \cdot S_k(t) = w + \Omega \cdot S_k(t)$.

Соответствующие модификации алгоритмов Эвристика 1 и Эвристика 2 (в случае учета затрат на прерывания) будем называть Эвристика П1 и Эвристика П2. Для каждого из 3-х вариантов механизма образования временных затрат на прерывания и восстановления работ были проведены исследования, начиная с какого отношения характерной длины I_i к величинам Ω и ω алгоритм Эвристика П1 становится более предпочтительным (то есть с большей вероятностью строящим допустимое расписание в случае его существования), чем алгоритм Эвристика П2. Выбирались величины $\gamma_1 = \dfrac{\Omega}{\langle I_i \rangle}$ для первого варианта, $\gamma_2 = \dfrac{w}{\langle I_i \rangle}$ для второго и обе эти величины для третьего варианта. Выходом каждого испытания является вектор **Q =** (q_1 q_2), компоненты которого могут принимать значения 0 или 1 (q_1 = 1, если Эвристика П1 построила допустимое расписание, 0 в противном случае; анало-

гично, $q_2 = 1$, если Эвристика П2 построила допустимое расписание, 0, если нет).

1.3.1. Испытания алгоритмов в случае первой модели образования временных затрат на прерывания

Параметр γ_1 меняется от 0 до 0.3 с шагом 0.01. Для каждого значения параметра проводится 6000 испытаний по вышеописанному сценарию. Затем сравнивается количество получившихся в результате этих испытаний векторов **Q** = (1 0), которое будем обозначать как Q_1, с количеством векторов **Q** = (0 1), которое обозначим как Q_2.

Таблица 5. Экспериментально полученные значения Q_1 и Q_2 в зависимости от γ_1.

γ_1	Q_1	Q_2
0	225	1004
0.01	222	343
0.02	216	208
0.03	202	153
0.04	195	112
0.05	183	110
0.06	170	78
0.07	169	76
0.08	168	72
0.09	164	65
0.10	166	52
0.11	165	56
0.12	151	43
0.13	154	43
0.14	134	47
0.15	123	48
0.16	111	30
0.17	109	36
0.18	108	43
0.19	119	28

0.20	87	30
0.21	110	35
0.22	80	34
0.23	81	22
0.24	75	37
0.25	81	34
0.26	67	19
0.27	63	33
0.28	62	23
0.29	44	32
0.30	59	18

На основании полученных данных можно проанализировать целесообразность применения алгоритмов Эвристика П1 или Эвристика П2 в зависимости от величины γ_1. Изобразим более наглядно полученные данные на графике в виде зависимостей $Q_1(\gamma_1)$ и $Q_2(\gamma_1)$ (рис. 5).

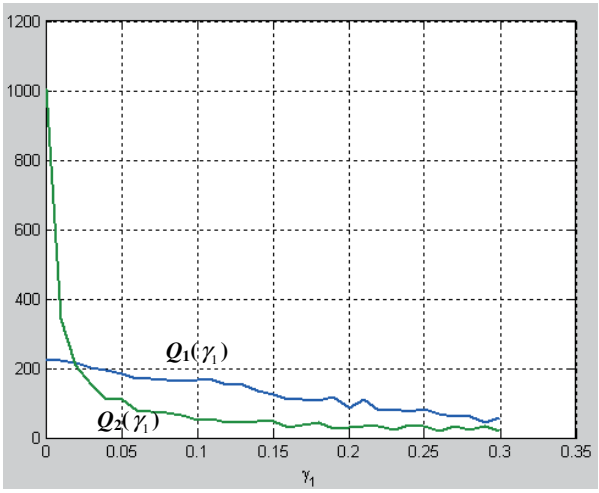


Рис. 5. Зависимости $Q_1(\gamma_1)$ и $Q_2(\gamma_1)$ в случае 1-го варианта модели образования временных затрат на прерывания и возобновления работ.

Точка пересечения полученных кривых и является той пограничной величиной γ_1, начиная с которой использование алгоритма Эвристика П1 становится предпочтительнее использования алгоритма Эвристика П2 при 1-м варианте модели образования дополнительных временных затрат на прерывания и возобновления работ. Полученное экспериментально такое пограничное значение γ_1 равно 0.02.

1.3.2. Испытания алгоритмов в случае второй модели образования временных затрат на прерывания

Параметр γ_2 меняется от 0 до 0.3 с шагом 0.01. Для каждого значения параметра проводится 6000 испытаний по вышеописанному сценарию. Затем сравнивается количество получившихся в результате этих испытаний векторов **Q** = (1 0), которое будем обозначать как Q_1, с количеством векторов **Q** = (0 1), которое обозначим как Q_2.

Таблица 6. Экспериментально полученные значения Q_1 и Q_2 в зависимости от γ_2.

γ_2	Q_1	Q_2
0	222	1039
0.01	202	618
0.02	220	442
0.03	223	341
0.04	211	285
0.05	192	266
0.06	204	202
0.07	189	186
0.08	211	158
0.09	202	182
0.10	204	134
0.11	181	156
0.12	188	116
0.13	182	111
0.14	178	103
0.15	168	121

0.16	176	108
0.17	167	107
0.18	165	111
0.19	170	97
0.20	175	107
0.21	165	78
0.22	155	94
0.23	159	67
0.24	175	83
0.25	157	81
0.26	167	86
0.27	143	65
0.28	150	64
0.29	162	59
0.30	152	84

Аналогично рассмотренным выше испытаниям алгоритмов в случае первой модели образования временных затрат на прерывания работ, можно проанализировать целесообразность применения алгоритмов Эвристика П1 или Эвристика П2 в зависимости от величины γ_2, изобразив полученные данные на графике в виде зависимостей $Q_1(\gamma_2)$ и $Q_2(\gamma_2)$ (рис. 6).

Точка пересечения полученных кривых и является той пограничной величиной γ_2, начиная с которой использование алгоритма Эвристика П1 становится предпочтительнее использования алгоритма Эвристика П2 при втором варианте механизма образования дополнительных временных затрат на прерывания и возобновления работ. Полученное экспериментально такое пограничное значение γ_2 равно 0.07.

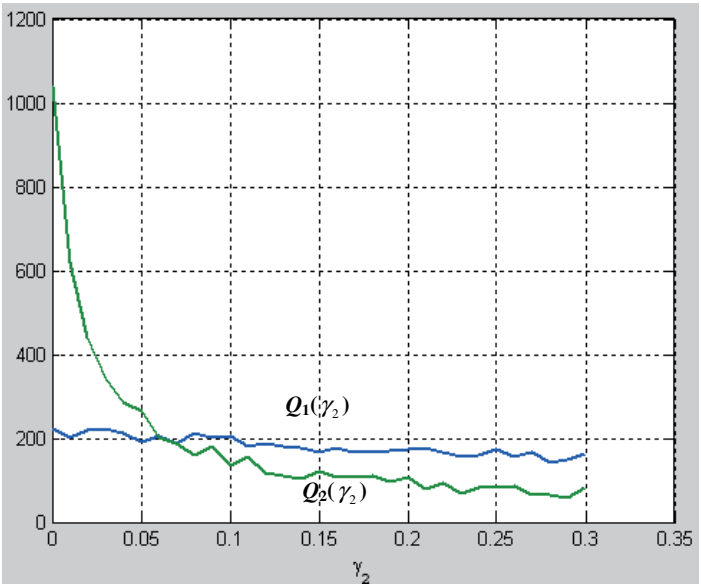


Рис 6. Зависимости $Q_1(\gamma_2)$ и $Q_2(\gamma_2)$ в случае 2-го варианта модели образования временных затрат на прерывания и возобновления работ.

1.3.3. Испытания алгоритмов в случае третьей модели образования временных затрат на прерывания

В отличие от первого и второго вариантов, в этом случае величина временных затрат на прерывания и восстановления работ на процессорах зависит сразу от двух параметров γ_1 и γ_2. Поэтому в данном случае рекомендации по целесообразности применения алгоритма Эвристика П1 или Эвристика П2 будут представлять из себя две области в положительном квадранте плоскости (γ_1, γ_2). Обозначим эти области как Γ_1 (область предпочтительного применения Эвристики П1) и Γ_2 (область предпочтительного применения Эвристики П2), и построим их при помощи предлагаемой ниже процедуры на основе экспериментальных данных.

Составим двумерный массив векторов**,** каждое из измерений которого представляет из себя различные значения параметров γ_1 и γ_2, а

каждый его элемент принимает значение «1» или «2» в зависимости от того, какой из алгоритмов является более предпочтительным для данной комбинации γ_1 и γ_2. Предпочтительность того или иного алгоритма выяснялась способом, аналогичном предложенному для анализа первых двух вариантов – проводилось 6000 испытаний для каждой пары параметров и сравнивались количества векторов **Q** = (1 0) и **Q** = (0 1) (соответственно, Q_1 и Q_2). Оба параметра меняются в пределах этого массива от 0 до 0.15 с шагом 0.005, то есть размерность массива равна 31×31. Полученные экспериментальные данные отражены на плоскости (γ_1, γ_2) на рис. 8. Светлый цвет соответствует элементам массива с номером «1» (предпочтительна Эвристика П1), темный – элементам с номером «2» (предпочтительна Эвристика П2) (рис. 7).

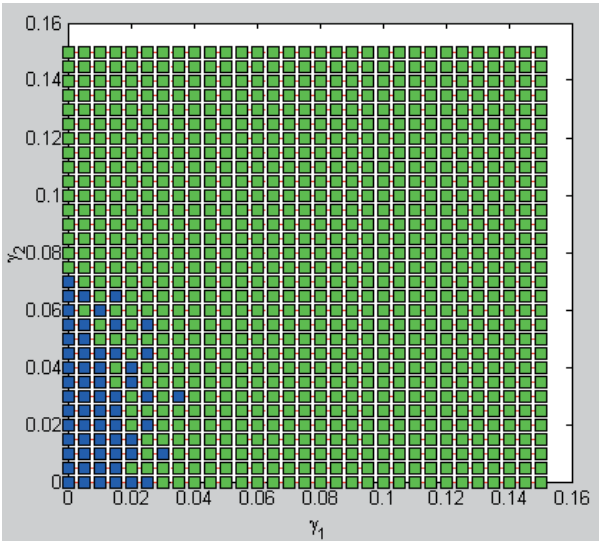


Рис 7. Области $Q_1 > Q_2$ и $Q_2 > Q_1$ на плоскости (γ_1, γ_2) в случае 3-го варианта модели образования временных затрат на прерывания и возобновления работ.

В следующих разделах данной главы рассматривается ряд еще более реалистичных математических моделей систем жесткого реального времени. Решается проблема распределения памяти, необходи-

мой для выполнения работ на процессорах, так как вся поступающая и обрабатываемая информация, как правило, не может полностью поместиться в памяти процессора, а следовательно, требуется планировать ее загрузку и выгрузку, которые также занимают определенное время. Также будет рассмотрен случай, когда граф связей процессоров в системе неполный, как это нередко бывает в реальных многопроцессорных системах ([42]).

1.4. Ограничения по памяти и скорости загрузки. Связи – полный граф. Общий директивный интервал

В предыдущих двух разделах настоящей главы рассматривались варианты исходной задачи о поиске допустимого расписания в многопроцессорной системе реального времени в случае отсутствия ограничений по объему памяти процессоров. В случае реальных систем, в которых объем памяти процессоров всегда конечен, это может быть истолковано как возможность заведомой загрузки полного объема данных для выполнения всех имеющихся работ в оперативную память каждого процессора. К сожалению, на практике это обычно не так, и поэтому в этой главе рассматриваются системы реального времени с ограничениями по памяти процессоров. Более того, загрузка данных, необходимых для обработки какой-либо работы, в память процессора является сама по себе требующим временных затрат процессом, в общем случае сравнимым по продолжительности с временем выполнения этой работы. Поэтому возникает задача построения допустимого расписания рассматриваемой системы реального времени не только как расписания выполнения работ на процессорах, но и расписания загрузки в процессоры необходимых для выполнения этих работ данных, т.е. фактически строится совокупность этих двух расписаний ([44],[45]).

1.4.1. Однопроцессорный случай

Начнем рассмотрение этой проблемы с более простого однопроцессорного случая, для решения которого предложим точный алгоритм ([43]). Дадим точную постановку задачи.

1.4.1.1. Постановка задачи и предварительные соображения

Имеется однопроцессорная система и n работ, которые нужно выполнить за время, не превосходящее $T_{\max}$. Работа i ($i = 1,\ldots,n$) имеет сложность p_i, которая определяется как время ее выполнения (так как процессор в рассматриваемой системе один, его скорость без ограничения общности можно считать единичной). До начала исполнения работы i в память процессора должны быть загружены соответствующие данные. Предполагается, что время загрузки данных для работы i прямо пропорционально загружаемому объему памяти, поэтому без ограничения общности можно считать, что оно равно их объему. Обозначим эту величину $t_i (i = 1,...,n)$. Объем памяти также удобно определить во временных единицах, будем считать, что всю память процессора можно полностью загрузить за время R. Память может быть загружена (частично или полностью) некоторыми данными уже в начальный момент времени. Удаление данных из памяти происходит мгновенно сразу после выполнения соответствующей работы. Считаем, что архитектура системы такова, что загрузка данных в память процессора и выполнение им работ – независимые процессы, которые могут идти параллельно, не влияя друг на друга. При выполнении работ и загрузке данных допускаются прерывания, требующие временных затрат λ. Необходимо определить, существует ли расписание, позволяющее выполнить все работы за время, не превосходящее $T_{\max}$, и в случае положительного ответа указать такое расписание.

Как уже было отмечено выше, допустимое расписание при такой постановке задачи фактически состоит из двух расписаний: собственно расписания выполнения работ на процессоре и расписания загрузки данных в память процессора. Ниже приведены несколько очевидных соображений, на которые мы будем ссылаться при дальнейших рассуждениях.

1) Для существования допустимого расписания необходимо выполнение следующих условий:

$$\sum_{i=1}^{n} p_i \le T_{\max}, \qquad (4)$$

т.е. суммарное время выполнения всех работ не превышает $T_{\max}$;

$$\max_i t_i \le R, \tag{5}$$

т.е. данные, необходимые для выполнения каждой работы, могут целиком поместиться в память;

$$\sum_{i=1}^{n} t_i - R \le T_{\max} - \min_i p_i, \tag{6}$$

т.е. суммарное время загрузки в память данных для всех работ не превышает времени, отведенного на это условиями задачи. Величина $-R$ в левой части неравенства (6) означает возможность полной загрузки памяти уже в нулевой момент времени, а стоящее справа выражение показывает, что загрузка последней задачи должна быть выполнена до начала ее выполнения, то есть до момента времени $T_{\max} - \min_i p_i$. Если хотя бы одно из условий (4) - (6) не выполнено, то допустимого расписания не существует. Поэтому первой стадией предложенного далее алгоритма является проверка каждого из этих неравенств.

2) Прерывания выполнения работ не оправданы, поскольку все работы имеют один и тот же директивный интервал, а прерывания какой-либо работы потребуют более продолжительного нахождения ее данных в памяти.

3) Последней должна выполняться работа с наименьшим временем исполнения p_i, поскольку в этом случае на загрузку данных всех работ будет оставаться максимальное время, равное $R + T_{\max} - p_{i_{\min}}$, где $p_{i_{\min}} = \min_i p_i$.

4) Так как у всех работ одинаковый директивный интервал (0; $T_{\max}$], то оптимальным порядком выполнения работ является тот, который обеспечивает минимальное суммарное время простоев процессора от выполнения работ. Такие простои могут происходить только тогда, когда закончено выполнение очередной работы, а данные

для выполнения следующей еще не успели загрузиться в память процессора.

1.4.1.2. Построение оптимального порядка выполнения работ

Опишем алгоритм построения оптимального порядка работ. Будем задавать его последовательностью $B=\{b_1,...,b_n\}$, каждый член которой задает номер соответствующей по порядку работы. Построим эту последовательность при помощи следующего алгоритма, состоящего из *n* шагов:

Шаг 1. Положить $b_n=i_{\min}; Q_n=t_{i_{\min}}$; $B=\{b_n\}$.

Шаг r. Пусть на предыдущих (*r* – 1) шагах определены величины $b_{n-r+2},...,b_n$, $Q_{n-r+2},...,Q_n$ и множество $B=\{b_{n-r+2},...,b_n\}$. Тогда, если существует такое $i\notin B$, что $p_i\le Q_{n-r+2}$, то положить $b_{n-r+1}=i_0$, $Q_{n-r+1}=Q_{n-r+2}-p_{b_{n-r+1}}+t_{b_{n-r+1}}$, где i_0 таково, что одновременно выполнены соотношения $p_{i_0}=\max\limits_{i\notin B} p_i$ и $p_{i_0}\le Q_{n-r+2}$, (т.е. среди пока не принадлежащих множеству *B* номеров ищем номер i_0 такой работы, которая имеет максимальную, но при этом не превышающую Q_{n-r+2} сложность p_{i_0}). Если же такого i не существует, то положить $b_{n-r+1}=i_1$, $Q_{n-r+1}=t_{b_{n-r+1}}$, где i_1 таково, что $p_{i_1}=\min\limits_{i\notin B} p_i$ (т.е. в этом случае просто ищем работу с минимальной сложностью, номер которой пока не принадлежит *B*). В обоих случаях в конце шага множество *B* принимает вид $B=\{b_{n-r+1},...,b_n\}$.

Обоснуем корректность предложенного алгоритма. Начальные значения переменных на Шаге 1 следуют из 3-го замечания разд. 1.4.1.1 Величина *Q*, вычисляемая на каждом шаге – это разница между временем начала загрузки и временем начала выполнения последней добавленной в множество *B* работы. Выбирая на каждом шаге очередную работу с временем выполнения, максимально приближающимся к предыдущему значению *Q*, но все еще меньшим его, мы достигаем максимально интенсивного использования времени, отведенного на загрузку данных в память, избегая перерывов в этом процессе. Если же таких работ не осталось, мы берем работу с наименьшим возмож-

ным значением p_i для того, чтобы минимизировать этот перерыв. Согласно четвертому замечанию, приведенному в разд. 1.4.1.1, порядок работ, обеспечивающий минимальное время простоя процессора, является оптимальным. Вычислительная сложность предложенного алгоритма $O(n^2 \log n)$ (число шагов равно *n*, а самой трудоемкой операцией на каждом шаге является сортировка оставшихся вне множества *B* работ).

Проиллюстрируем работу алгоритма построения оптимального порядка работ на конкретном примере. Пусть $n = 7$; $T_{\max} = 40$; $R = 8$; $p_1 = 1$, $t_1 = 3$; $p_2 = 5$, $t_2 = 4$; $p_3 = 7$, $t_3 = 8$; $p_4 = 9$, $t_4 = 7$; $p_5 = 4$, $t_5 = 6$; $p_6 = 8$, $t_6 = 2$; $p_7 = 5$, $t_7 = 8$.

Шаг 1. Самой низкой сложностью обладает 1-я работа: $b_7 = 1$; $Q_7 = 3$; $B = \{1\}$.

Шаг 2. Такого $i \notin B$, что $p_i \le Q_7 = 3$ не существует, поэтому $\min\limits_{i \notin B} p_i = 4 = p_5$, отсюда $i_1 = 5$. Поэтому $b_6 = 5$; $Q_6 = t_5 = 6$; $B = \{5,1\}$.

Шаг 3. Существует $i \notin B$ такое, что $p_i \le Q_6 = 6$, поэтому находим $p_{i_0} = 5$, $i_0 = 7$. Отсюда $b_5 = 7$; $Q_5 = Q_6 - p_5 + t_5 = 6 - 5 + 8 = 9$; $B = \{7,5,1\}$.

Шаг 4. Существует $i \notin B$ такое, что $p_i \le Q_5 = 9$, поэтому находим $p_{i_0} = 9$, $i_0 = 4$. Отсюда $b_4 = 4$; $Q_4 = Q_5 - p_4 + t_4 = 9 - 9 + 7 = 7$; $B = \{4,7,5,1\}$.

Шаг 5. Существует $i \notin B$ такое, что $p_i \le Q_4 = 7$, поэтому находим $p_{i_0} = 7$, $i_0 = 3$. Отсюда $b_3 = 3$; $Q_3 = Q_4 - p_3 + t_3 = 7 - 7 + 8 = 8$; $B = \{3,4,7,5,1\}$.

Шаг 6. Существует $i \notin B$ такое, что $p_i \le Q_3 = 8$, поэтому находим $p_{i_0} = 8$, $i_0 = 6$. Отсюда $b_2 = 6$; $Q_2 = Q_3 - p_6 + t_6 = 8 - 8 + 2 = 2$; $B = \{6,3,4,7,5,1\}$.

Шаг 7. Такого $i \notin B$, что $p_i \le Q_2 = 2$ не существует, поэтому $\min\limits_{i \notin B} p_i = 5 = p_2$ (осталась единственная работа, пока не принадлежащая *B*), отсюда $i_1 = 2$. Поэтому $b_1 = 2$; $Q_1 = t_2 = 4$; $B = \{2,6,3,4,7,5,1\}$. Таким образом, в результате работы алгоритма получили искомый оптимальный порядок выполнения заданных работ - {2,6,3,4,7,5,1}.

1.4.1.3. Алгоритм построения однопроцессорного расписания

Перенумеруем все работы в соответствии с найденным в пункте 1.4.1.2 оптимальным порядком B. Пусть T_i - момент окончания выполнения работы i (i = 1, …, n). Поскольку в силу второго замечания пункта 1.4.1.1 прерывания в оптимальном допустимом расписании в рассматриваемом случае отсутствуют, то величины T_i полностью определяют расписание выполнения работ. Временные интервалы загрузок данных в память будем задавать множеством $C = \{C_1, \ldots, C_n\}$, где каждое C_i (i = 1,…, n) состоит из одной или нескольких пар вида $\left\{C_i^{2j+1}, C_i^{2j+2}\right\}$, элементы которых обозначают соответственно момент начала и окончания j-го интервала загрузки данных для работы i (данные могут загружаться с прерываниями, за несколько интервалов, j = 0, 1, 2... – номера интервалов загрузки данных для работы i). Пусть t_i^0 - объем недогруженных данных для работы i к моменту времени $T_{i\text{-}1}$. Опишем алгоритм построения допустимого расписания.

Процедура 1.

1. Присвоить следующие начальные значения определенным выше величинам: $T_i = \sum_{j=1}^{i} p_j$, $T_0 = 0$, $T_{-1} = -R$, $C_i = \varnothing$, $t_i^0 = t_i$, $i = 1,\ldots,n$.

Процедура 2.

2. Для $r = n$ - 1, ..., 0 выполнить: если $(T_r - T_{r-1}) - t_{r+1} > 0$ то включить в C_r пару $\{T_r - t_{r+1}, T_r\}$ и положить $t_{r+1}^0 = 0$; если $(T_r - T_{r-1}) - t_{r+1} \le 0$, то включить в C_r пару $\{T_{r\text{-}1}, T_r\}$ и положить $t_{r+1}^0 = t_{r+1} - (T_r - T_{r-1})$.

Таким образом, после второй процедуры алгоритма будут определены множество C и величины t_i^0, i = 1, …, n.

Процедура 3.

3. Для $r = 1, \ldots, n$ выполнить:

3.1. Если $C_r^1 > T_{r-2}$, то:

3.1.1. положить $\Delta_1 = T_{r-2}; \Delta_2 = C_r^1$;

3.1.2. Обозначим $r_1 = r+1$. Если $t^0_{r_1} = 0$, то перейти на следующую итерацию шага 3. Если $t^0_{r_1} > 0$, то перейти на 3.1.3.

3.1.3. если $t^0_{r_1} \le \Delta_2 - \Delta_1$, то:

3.1.3.1. включить в C_{r_1} пару $\{\Delta_1, \Delta_1 + t^0_{r_1}\}$;

3.1.3.2. положить $\Delta_1 = T_{r-2} + t^0_{r_1}$, $t^0_{r_1} = 0$;

3.1.3.3. перейти на шаг 3.1.2;

3.1.4. если $t^0_{r_1} > \Delta_2 - \Delta_1$, то:

3.1.4.1. уменьшить текущее значение $t^0_{r_1}$ на $\Delta_2 - \Delta_1$;

3.1.4.2. включить в C_{r_1} пару $\{\Delta_1, \Delta_2\}$;

3.1.4.3. перейти на следующую итерацию Процедуры 3;

3.2. Если $C^1_r \le T_{r-2}$, то:

3.2.1. увеличить текущее значение C^2_r на t^0_r;

3.2.2. для всех $r_1 \ge r - 1$ увеличить текущие значения T_{r_1} на t^0_r. Если при этом получили, что $T_n > T_{\max}$, то допустимого расписания не существует и выполнение алгоритма можно закончить;

3.2.3. положить $t^0_r = 0$;

3.2.4. увеличить на t^0_r все текущие значения C^j_i для всех i, j таких, что $C^j_i > C^2_r$;

3.2.5. перейти на следующий шаг Процедуры 3.

Для большей наглядности проиллюстрируем работу вышеизложенного алгоритма, продолжив рассмотрение конкретной задачи из предыдущего раздела, для которой мы уже нашли оптимальный порядок и перенумеровали все работы в соответствии с ним. Теперь $p_1 = 5$,

$t_1 = 4$; $p_2 = 8$, $t_2 = 2$; $p_3 = 7$, $t_3 = 8$; $p_4 = 9$, $t_4 = 7$; $p_5 = 5$, $t_5 = 8$; $p_6 = 4$, $t_6 = 6$; $p_7 = 1$, $t_7 = 3$. Напоминаем, что $n = 7$; $T_{max} = 40$; $R = 8$.

1. При инициализации получаем $T_{-1} = -8$; $T_0 = 0$; $T_1 = 5$; $T_2 = 13$; $T_3 = 20$; $T_4 = 29$; $T_5 = 34$; $T_6 = 38$; $T_7 = 39$.

2. Подставляя в процедуру второго шага алгоритма конкретные значения T_r и t_r, получаем: $C_6 = \{35, 38\}$, $t_7^0 = 0$; $C_5 = \{29, 34\}$, $t_6^0 = 1$; $C_4 = \{21, 29\}$, $t_5^0 = 0$; $C_3 = \{13, 20\}$, $t_4^0 = 0$; $C_2 = \{5, 13\}$, $t_3^0 = 0$; $C_1 = \{3, 5\}$, $t_2^0 = 0$; $C_0 = \{-4, 0\}$, $t_1^0 = 0$.

3. Теперь, применяя процедуру третьего пункта алгоритма (т.е. совершая прямой проход во времени), проверим, существует ли при заданных условиях задачи допустимое расписание, и если да, то зададим его окончательными значениями T_i (концы выполнения работ) и множеством C интервалов загрузок данных в память. В результате этого прохода условие пункта 3.2 оказалось истинным только на одном шаге, а именно на 6-м ($r = 6$), поэтому там произошел сдвиг выполнения и загрузки 6-го и 7-го заданий на величину $t_6^0 = 1$. Так как $T_n = T_7 = 40 \le T_{max} = 40$, то допустимое расписание существует. Оно изображено на рис. 8. Жирными цифрами на этом рисунке проставлены исходные номера работ, до перенумерования их в соответствии с найденным оптимальным порядком.

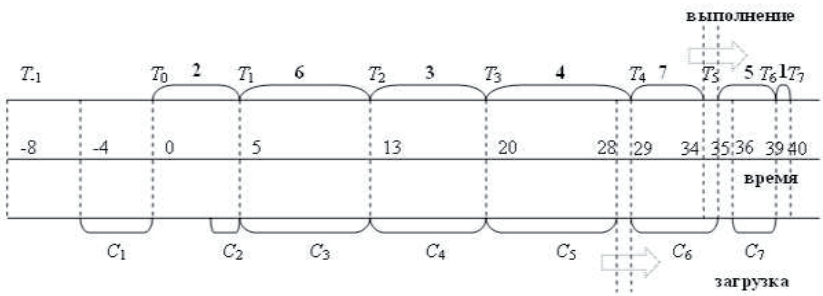


Рис. 8. Допустимое расписание, построенное в результате работы алгоритма. Стрелками показан сдвиг, произошедший на шаге 3.2 прямого прохода алгоритма относительно расписания, полученного при обратном проходе алгоритма (Процедура 2).

Обоснование предложенного алгоритма дает следующая теорема.

Теорема 1. Построенный алгоритм всегда корректно решает задачу поиска допустимого расписания в условиях разд. 1.4.1.1 и обладает вычислительной сложностью $O\left(n^2 \log n\right)$.

1.4.2. Многопроцессорный случай

Итак, только что в разд. 1.4.1 был построен точный алгоритм, решающий задачу поиска допустимого расписания в однопроцессорной системе жесткого реального времени в случае заданного ограничения на память процессора и одинаковых директивных интервалов всех работ, показана его корректность, и найдена его вычислительная сложность. Предлагаемый ниже алгоритм является обобщением этого точного алгоритма на многопроцессорный случай ([48]). Процессоры в рассматриваемой системе реального времени могут быть разными по скорости, при этом считаем, что каждый процессор характеризуется, помимо объема памяти, двумя скоростями: скоростью выполнения работ и скоростью загрузки данных в свою память. Относительно возможности прерываний работ будут рассмотрены два варианта – либо они разрешены, либо запрещены, однако следует отметить, что алгоритм в любом случае генерирует расписание без прерываний. Алгоритм является эвристическим, так как были построены случаи, когда он работает некорректно. Тем не менее, вследствие NP-трудности обоих вариантов эвристический подход представляется наиболее эффективным для применения на практике. Приведем точную постановку многопроцессорного случая рассматриваемого в данном разделе случая задачи о построении допустимого расписания.

1.4.2.1. Постановка задачи

Имеется многопроцессорная система, состоящая из m процессоров, характеризующихся скоростями выполнения работ s_j ($j = 1,\ldots,m$), скоростями загрузки данных l_j ($j = 1,\ldots,m$) и объемами памяти V_j ($j = 1,\ldots, m$). Имеется n работ, которые нужно выполнить за время, не превосходящее $T_{\max}$. Работа i ($i = 1,\ldots, n$) имеет сложность

p_i (i = 1,…, n), это обозначает, что время ее выполнения на j-м процессоре равно $\frac{p_i}{s_j}$, и объем данных v_i ($i = 1,..., n$), что обозначает, что время загрузки ее данных на j-й процессор равно $\frac{v_i}{l_j}$. До начала выполнения работы i в память процессора j, на котором она будет выполнена, должны быть загружены соответствующие этой работе данные. Загрузка данных может производиться на каждом процессоре одновременно с выполнением какого-либо задания. Память может быть загружена (частично или полностью) некоторыми данными уже в начальный момент времени. Удаление данных из памяти происходит мгновенно сразу после выполнения соответствующей работы. Прерывания работ, если они допустимы, не требуют никаких дополнительных затрат процессорного времени, кроме повторной загрузки данных в память процессора, на котором возобновляется работа, если до этого их там не было. Граф связей между процессорами полный. Необходимо определить, существует ли расписание, позволяющее выполнить все работы за время, не превосходящее $T_{\max}$, и в случае положительного ответа указать такое расписание. Без ограничения общности можно считать все параметры задачи натуральными числами.

Допустимое расписание при такой постановке задачи фактически состоит из двух расписаний: собственно расписания выполнения работ на процессоре и расписания загрузки данных в память процессора.

1.4.2.2. NP-трудность

Лемма 5. Сформулированная в разд. 1.4.2.1 задача о допустимом многопроцессорном расписании, в котором прерывания запрещены, принадлежит классу NP-трудных задач.

Перейдем теперь к случаю, когда прерывания допустимы.

Лемма 6. Сформулированная в разд.1.4.2.1 задача о допустимом многопроцессорном расписании, в котором прерывания разрешены, принадлежит классу NP-трудных задач.

Таким образом, мы показали, что оба варианта рассматриваемой задачи о поиске многопроцессорного допустимого расписания с огра-

ничениями по объему памяти принадлежат к классу NP-трудных задач.

1.4.2.3. Эвристический алгоритм

Аналогично алгоритму для однопроцессорного случая, построенному в разд. 1.4.1, предлагаемый алгоритм состоит из двух частей. В первой определяется распределение работ по процессорам и порядок их выполнения на каждом из них, во второй для каждого процессора строится допустимое расписание или доказывается, что его не существует. Рассмотрим первую часть предлагаемого эвристического алгоритма.

1.4.2.3.1. Распределение работ по процессорам и определение порядка выполнения работ на каждом из них

Опишем алгоритм построения распределения работ по процессорам и определения порядка их выполнения на каждом процессоре. Будем задавать это упорядоченное распределение множеством последовательностей *B*, состоящим из *m* последовательностей $B^j = \left\{b_1^j,...,b_{n_j}^j\right\}$, $j = 1,...,m$, каждый член которой задает номер соответствующей по порядку работы на *j*-м процессоре. При этом n_j - число работ, назначенных на *j*-й процессор. Без ограничения общности можно полагать, что процессоры упорядочены по своим вычислительным мощностям (скоростям выполнения работ) от самого медленного до самого производительного.

Построим указанное множество последовательностей при помощи следующего алгоритма, состоящего из *n* шагов.

Первые *m* шагов мы назначаем самые несложные (в смысле времени выполнения) *m* заданий из имеющегося набора в качестве последних выполняемых на каждом процессоре. При этом следим, чтобы уже на этих первых шагах мы не нарушили ограничений по памяти процессоров и по возможности выполнения работ за заданный директивный временной интервал (вполне допустима ситуация, когда какие-либо из работ либо не могут быть выполнены на каком-либо не самом производительном процессоре системы за отведенный дирек-

тивный интервал, либо их данные не могут поместиться в какой-либо не самый вместительный в смысле оперативной памяти процессор). Рассуждения, почему это выгодно с точки зрения построения оптимального расписания, в точности такие же, как в однопроцессорном случае, описанном в разд. 1.4.1. Также обращаем внимание на то, что, так как в этом случае мы заранее не знаем, какая по счету работа будет последней на каждом процессоре (т.е. числа n_j, $j = 1,..., m$), мы назначаем эти работы на первую позицию в множествах B^j, а уже в процессе выполнения алгоритма, добавляя в эти множества новые элементы, мы будем добавлять их на первую позицию, увеличивая индекс уже имеющихся элементов на единицу. Дадим формальное описание каждого из *m* первых шагов:

Шаг *j*. (*j* = 1,…, *m*). Положить $b_1^j = i_0^j$, где i_0^j такое, что $p_{i_0^j} = \min\limits_{i \notin B} p_i$. Присвоить значения: $Q_1^j = \dfrac{v_{i_0^j}}{l_j}$, $B^j = \{b_1^j\}$, $B = \bigcup\limits_{i=1}^{j} B^i$, $n_j = 1$, $R^j = T_{\max} - \dfrac{p_{i_0^j}}{s_j}$ (смысл переменных Q_1^j и R^j см. далее по тексту). При этом следим, чтобы выполнялись ограничения $\dfrac{p_{i_0^j}}{s_j} \le T_{\max}$ (ограничение по времени выполнения) и $v_{i_0^j} \le V_j$ (ограничение по объему памяти) для каждой назначаемой на этом шаге работы. Если на каком-то из рассматриваемых первых *m* шагов эти ограничения не выполнены, то мы либо пропускаем текущий процессор и переходим к следующему по порядку, либо, если таких процессоров больше нет, делаем вывод о том, что допустимого расписания не существует.

Идея остальных шагов заключается в том, чтобы минимизировать время простоя процессора как на загрузке, так и на выполнении работ. Для этого на каждом последующем шаге выбирается такая работа из оставшихся, и назначается на такой процессор, чтобы разница между временем загрузки данных последней назначенной на этот процессор работы (по времени первой, так как мы назначаем работы двигаясь с конца директивного интервала в его начало) и временем выполнения

этой работы на данном процессоре была минимальной. При этом мы также следим, чтобы на каждом из этих процессоров еще существовал свободный отрезок директивного интервала достаточной длины, чтобы назначить туда эту работу на выполнение. Как и в рассмотренном в разд. 1.4.1 однопроцессорном случае, считаем, что данные могут быть загружены в память еще до начала директивного интервала.

Шаг *r*. ($r = m+1,\ldots, n$). На предыдущих ($r - 1$) шагах были определены следующие величины:

n_j, $j = 1,...,m$ – количество уже назначенных работ на каждый *j*-й процессор.

$B^j = \{b_1^j,...,b_{n_j}^j\}$, $j = 1,...,m$ - порядок индексов уже назначенных работ на каждом *j*-м процессоре.

$Q_1^j,...,Q_{n_j}^j$, $j = 1,...,m$ - время загрузки данных последней назначенной на процессор *j* работы. Заметим, что если смотреть в хронологическом порядке с точки зрения выполнения работ на процессоре, то она наоборот - первая из назначенных. Дело в том, что при работе алгоритма мы двигаемся по времени назад.

R^j, $j = 1,...,\ m$ - оставшееся от исходной длины директивного интервала свободное процессорное время на *j*-м процессоре.

Далее, ищем такие значения *j* и i_0^r, что выражение $\left| Q_1^j - \frac{p_{i_0^r}}{s_j} \right|$ имеет минимальное значение при следующих ограничениях:

1) $j = 1,...,\ m$ (рассматриваются все процессоры).

2) $i_0^r \notin B$ (рассматриваются только еще не назначенные работы).

3) $\frac{p_{i_0^r}}{s_j} \le R^j$ (процессор *j* должен иметь необходимое количество свободного процессорного времени на выполнение данной работы).

4) $v_{i_0^r} \le V_j$ (*j*-й процессор должен обладать достаточным объемом памяти для загрузки данных для выполнения данной работы).

Пусть найдены искомые значения *j* и i_o^r. Тогда выполняем следующие операции присваивания значений переменным (здесь и далее

символом «$\leftarrow$» обозначена операция присваивания переменной в левой части выражения значения его правой части):

$$b_{k+1}^j \leftarrow b_k^j,\ k = n_j, ..., 1$$

$$Q_{k+1}^j \leftarrow Q_k^j,\ k = n_j, ..., 1$$

$$n_j \leftarrow n_j + 1$$

$$b_1^j \leftarrow i_0^r$$

Если $Q_1^j \geq \frac{p_{i_0^r}}{s_j}$, то $Q_1^j \leftarrow Q_2^j - \frac{p_{i_0^r}}{s_j} + \frac{v_{i_0^r}}{l_j}$

Если $Q_1^j < \frac{p_{i_0^r}}{s_j}$, то $Q_1^j \leftarrow \frac{v_{i_0^r}}{l_j}$

$$R^j \leftarrow R^j - \frac{p_{i_0^r}}{s_j}$$

Переходим на следующий шаг.

Если на каком-то шаге $r < n$ оказалось, что сформулированная выше экстремальная задача с ограничениями не имеет решения, то это означает отсутствие возможности построения допустимого расписания данным алгоритмом. В противном случае мы получаем искомое разбиение работ по процессорам с указанным порядком выполнения.

Вычислительная сложность предложенного алгоритма $O(n^2 m \log n)$ (число шагов равно *n*, а самой трудоемкой операцией на каждом шаге является решение специфической комбинаторной экстремальной задачи с двумя переменными сложностью порядка $O(nm \log n)$).

1.4.2.3.2. Построение допустимого расписания

Как уже отмечалось, эта часть предлагаемого алгоритма полностью аналогична точному однопроцессорному алгоритму, изложенному в разд. 1.4.1, примененному для каждого из m процессоров, относительно назначенных на него в предыдущей части алгоритма работ и их установленного порядка. Невозможность построить расписание для одного из этих процессоров путем применения описанной в разд.

1.4.1 процедуры будет означать отсутствие допустимого расписания для всей задачи в целом.

Легко видеть, что вычислительная сложность всего предложенного алгоритма поиска допустимого расписания в случае многопроцессорной системы с ограничениями по памяти процессоров и одинаковыми директивными интервалами всех работ составляет $O(n^2 m \log n)$.

1.4.2.4. Анализ предложенного алгоритма

Как уже говорилось вначале разд. 1.4.2, в расписаниях, генерируемых предложенным алгоритмом, согласно его определению, не встречается прерываний и восстановлений работ. Тем не менее, в этом параграфе мы будем анализировать корректность его работы как в случае, когда прерывания невозможны, так и в случае, когда они допускаются условиями задачи.

Предложенный алгоритм является эвристическим, так как для каждого из двух случаев наличия или отсутствия возможности прерываний работ можно привести пример его некорректной работы. Напоминаем, что под некорректной работой эвристического алгоритма поиска допустимого расписания подразумевается его отрицательный результат при некотором наборе значений параметров, которыми задаются условия задачи, в то время как на самом деле расписание при этих параметрах существует.

Построим такие примеры для обоих рассматриваемых случаев.

Сначала рассмотрим случай, когда прерывания допускаются условиями задачи. Для простоты будем считать, что временные затраты процессоров на прерывания и возобновления работ пренебрежимо малы. Из общих соображений понятно, что в случае, когда перед выполнением задачи ее данные нужно загрузить в память и это занимает какое-то время, прерывания и переключения не выгодны в том смысле, что нам придется загружать одни и те же данные, то есть расходовать процессорное время на загрузку, два и более раз вместо одного. Тем не менее, в многопроцессорном случае (в отличие от рассмотренного в разд. 1.4.1 однопроцессорного, в котором аналогичные рассуждения

приводили к точному алгоритму решения задачи о поиске допустимого расписания) достаточно просто построить искомый пример некорректной работы алгоритма. Пусть заданы следующие параметры задачи: $m = 2$ (число процессоров); $s_1 = 1$; $s_2 = 1$ (производительности процессоров); $V_1 = 2$; $V_2 = 2$ (объемы памяти процессоров); $l_1 = 1$; $l_2 = 1$ (скорости загрузки данных процессоров); $n = 3$ (число работ); $p_1 = 3$; $p_2 = 4$; $p_3 = 3$ (вычислительные сложности работ); $v_1 = 2$; $v_2 = 2$; $v_3 = 2$ (объемы данных работ); $T_{\max} = 5$ (длина общего директивного интервала).

Посмотрим, каким образом на данном наборе параметров задачи будет действовать рассматриваемый алгоритм:

Шаг 1. $i_0^1 = 1$, т.к. $p_1 = \min\limits_{i \notin \varnothing} p_i$; $b_1^1 = i_0^1 = 1$.

$$Q_1^1 = \frac{v_{i_0^1}}{l_1} = \frac{v_1}{l_1} = \frac{2}{1} = 2;\ B^1 = \{b_1^1\} = \{1\},\ B = \{1\},\ n_1 = 1.$$

$$R^1 = T_{\max} - \frac{p_{i_0^1}}{s_1} = T_{\max} - \frac{p_1}{s_1} = 5 - \frac{3}{1} = 2.$$

Ограничения $\dfrac{p_{i_0^1}}{s_1} \le T_{\max}$ $\left(\dfrac{3}{1} \le 5\right)$ и $v_{i_0^1} \le V_1$ $(2 \le 2)$ выполнены.

Шаг 2. $i_0^2 = 3$, т.к. $p_3 = \min\limits_{i \notin \{1\}} p_i$; $b_1^2 = i_0^2 = 3$.

$$Q_1^2 = \frac{v_{i_0^2}}{l_2} = \frac{v_3}{l_2} = \frac{2}{1} = 2;\ B^2 = \{b_1^2\} = \{3\},\ B = \{1,3\},$$

$$n_2 = 1. \qquad R^2 = T_{\max} - \frac{p_{i_0^2}}{s_2} = T_{\max} - \frac{p_3}{s_2} = 5 - \frac{3}{1} = 2.$$

Ограничения $\dfrac{p_{i_0^2}}{s_2} \le T_{\max}$ $\left(\dfrac{3}{1} \le 5\right)$ и $v_{i_0^2} \le V_1$ $(2 \le 2)$ выполнены.

Шаг 3. Так как у нас осталась последняя не назначенная работа (вторая по порядку, $p_2 = 4$, $t_2 = 2$), то $i_0^3 = 2$, т.е. $p_{i_0^3} = 4$.

Рассматривая ограничение $\dfrac{p_{i_0^3}}{s_j} \le R^j$ для обоих имеющихся в распоряжении процессоров, получаем, что, т.к. $R^1 = R^2 = 2$ и $s_1 = s_2 = 1$,

$\frac{4}{1} > 2$ и ограничение не выполнено. Таким образом, алгоритм делает вывод о том, что допустимого расписания не существует.

С другой стороны, легко видеть, что расписание на самом деле существует и оно почти очевидно, просто рассматриваемый алгоритм не смог его построить из-за своей ограниченности на построении расписаний без прерываний. Действительно, одно из допустимых расписаний при заданных условиях задачи содержит прерывание второй работы и имеет следующий вид (для наглядности изобразим в виде диаграммы Гантта, сверху от оси каждого процессора показано выполнение работ, снизу - загрузка данных):

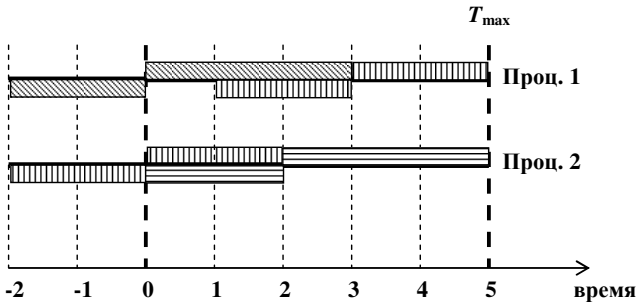


Рис 9. Пример допустимого расписания с прерываниями, являющегося решением на приводящем к отрицательному результату работы эвристического алгоритма наборе параметров.

На рис. 9 разными типами штриховки обозначены разные работы, тип штриховки выполнения работы и загрузки соответствующих ей данных в память процессора совпадают. В построенном допустимом расписании вторая работа, обозначенная вертикальной штриховкой, выполняется сначала на втором процессоре, затем прерывается и возобновляется на первом процессоре. При этом, как видно из диаграммы, в представленном допустимом расписании ее данные полностью загружаются в память два раза: сначала второго, затем первого процессора. Тем не менее, такой, на первый взгляд, неоптимальный путь построения расписания приводит в данном искусственно построенном случае к лучшему результату, чем не допускающий прерываний и повторных загрузок эвристический алгоритм.

Рассмотрим теперь случай, когда прерывания и возобновления работ недопустимы. И в этом случае можно построить пример, когда предложенный алгоритм работает некорректно. Пусть заданы следующие параметры задачи о поиске допустимого расписания: $m = 2$ (число процессоров); $s_1 = 1$; $s_2 = 1$ (производительности процессоров); $V_1 = 2$; $V_2 = 2$ (объемы памяти процессоров); $l_1 = 1$; $l_2 = 1$ (скорости загрузки данных процессоров); $n = 3$ (число работ); $p_1 = 1$; $p_2 = 2$; $p_3 = 3$ (вычислительные сложности работ); $v_1 = 2$; $v_2 = 2$; $v_3 = 2$ (объемы данных работ); $T_{\max} = 3$ (длина общего директивного интервала).

Посмотрим, каким образом на данном наборе параметров задачи будет действовать рассматриваемый алгоритм.

Шаг 1. $i_0^1 = 1$, т.к. $p_1 = \min\limits_{i \notin \varnothing} p_i$; $b_1^1 = i_0^1 = 1$.

$$Q_1^1 = \frac{v_{i_0^1}}{l_1} = \frac{v_1}{l_1} = \frac{2}{1} = 2;\ B^1 = \{b_1^1\} = \{1\},\ B = \{1\},\ n_1 = 1.$$

$$R^1 = T_{\max} - \frac{p_{i_0^1}}{s_1} = T_{\max} - \frac{p_1}{s_1} = 3 - \frac{1}{1} = 2.$$

Ограничения $\dfrac{p_{i_0^1}}{s_1} \le T_{\max}$ $\left(\dfrac{1}{1} \le 3\right)$ и $v_{i_0^1} \le V_1$ $(2 \le 2)$ выполнены.

Шаг 2. $i_0^2 = 2$, т.к. $p_2 = \min\limits_{i \notin \{1\}} p_i$; $b_1^2 = i_0^2 = 2$.

$$Q_1^2 = \frac{v_{i_0^2}}{l_2} = \frac{v_2}{l_2} = \frac{2}{1} = 2;\ B^2 = \{b_1^2\} = \{2\},\ B = \{1,2\},$$

$$n_2 = 1. \qquad R^2 = T_{\max} - \frac{p_{i_0^2}}{s_2} = T_{\max} - \frac{p_2}{s_2} = 3 - \frac{2}{1} = 1.$$

Ограничения $\dfrac{p_{i_0^2}}{s_2} \le T_{\max}$ $\left(\dfrac{2}{1} \le 3\right)$ и $v_{i_0^2} \le V_1$ $(2 \le 2)$ выполнены.

Шаг 3. Так как у нас осталась последняя не назначенная работа (третья по порядку, $p_3 = 3$, $t_3 = 2$), то $i_0^3 = 3$, т.е. $p_{i_0^3} = 3$.

Рассматривая ограничение $\dfrac{p_{i_0^3}}{s_j} \le R^j$ для обоих имеющихся в распоряжении процессоров получаем, что, т.к. $R^1 = 2$, $R^2 = 1$ и $s_1 = s_2 = 1$,

$\frac{3}{1} > 2$, $\frac{3}{1} > 1$ и ограничения на обоих процессорах не выполнены. Отсюда алгоритм делает вывод о том, что допустимого расписания не существует.

Тем не менее, при заданных значениях параметров допустимое расписание существует (см. рис.10).

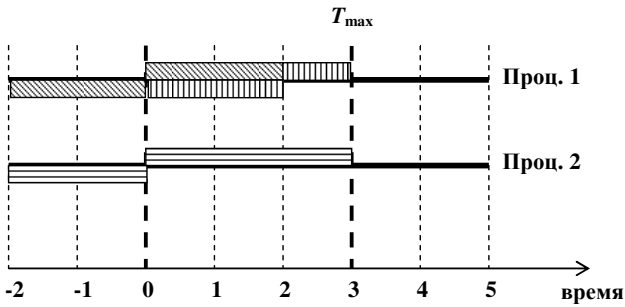


Рис 10. Пример допустимого расписания без прерываний, являющегося решением на приводящем к отрицательному результату работы эвристического алгоритма наборе параметров.

Проанализировав особенности двух представленных примеров некорректной работы предложенного эвристического алгоритма можно выдвинуть гипотезу о допустимых границах его применения. Нетрудно заметить, что в обоих случаях некорректность поведения алгоритма была связана с его обработкой работы, время выполнения процессором которой было сравнима и даже приближалось к длине директивного интервала $T_{\max}$ (вторая работа в первом примере и третья работа во втором примере). Отсюда делаем предположение о том, что чем меньше затрачиваемое процессорами время на обработку работ и загрузку их данных относительно длины директивного интервала, тем более корректно будет работать рассматриваемый эвристический алгоритм.

Для проверки этой гипотезы был проведен ряд серий сравнительных численных экспериментов при различных наборах параметров задачи. Отметим, что численные эксперименты проводились только для одного из случаев рассматриваемой задачи, когда прерывания в до-

пустимом расписании запрещены, в силу того, что в другом случае, когда прерывания допускаются, не удалось подобрать эталонного точного алгоритма, работающего разумное с практической точки зрения время, с результатами работы которого на заданном наборе параметров сравниваются результаты работы эвристического алгоритма и делается вывод о корректности его применения. В качестве точного алгоритма в случае, когда прерывания работ запрещены, применялась следующая процедура: анализировались все возможные распределения n работ по m имеющимся процессорам (всего m^n вариантов), затем для каждого такого распределения на каждом процессоре применялся точный алгоритм для однопроцессорного случая, описанный в разд.1.4.1 (вычислительная сложность $O\left(n^2 \log n\right)$).

Доказательство того, построенный таким образом алгоритм всегда находит точное решение задачи о допустимом расписании почти очевидно и заключается в том, что, руководствуясь сформулированной выше процедурой, мы перебираем все возможные случаи распределения работ по всем имеющимся процессорам, и для каждого такого случая применяем известный точный алгоритм, поэтому, так как прерывания и переключения работ с процессора на процессор недопустимы, мы не можем пропустить допустимое расписание, если оно действительно существует при заданном наборе параметров. Так как вычислительная сложность такого алгоритма довольно высока и составляет $O\left(m^{n+1} n^2 \log n\right)$, для оптимизации его работы при практическом применении для каждого распределения, прежде чем переходить собственно к однопроцессорному алгоритму, для каждого набора задач на каждом процессоре сначала проверялись следующие очевидные ограничения (введены следующие обозначения: K – множество работ, которые при данном распределении были назначены на рассматриваемый процессор, j – номер рассматриваемого процессора):

1) $\sum_{i \in K} p_i \le T_{\max}$, т.е. суммарное время выполнения всех назначенных данным распределением на рассматриваемый процессор работ не превышает $T_{\max}$;

2) $\max\limits_{i\in K} v_i \le V_j$, т.е. данные, необходимые для выполнения каждой из назначенных данным распределением на рассматриваемый процессор *j* работы, могут целиком поместиться в его память;

3) $\frac{1}{l_j}\left(\sum\limits_{i\in K} v_i - V_j\right) \le T_{\max} - \frac{\min\limits_{i\in K} p_i}{s_j}$, т.е. суммарное время загрузки в память данных для всех назначенных данным распределением на рассматриваемый процессор *j* работ не превышает времени, отведенного на это условиями задачи. Величина $-\frac{V_j}{l_j}$ в левой части неравенства означает возможность полной загрузки памяти уже в нулевой момент времени, а стоящее справа выражение показывает, что загрузка последней задачи (согласно точному алгоритму – это задача из множества *K* с минимальной вычислительной сложностью) должна быть выполнена до начала ее выполнения, то есть до момента времени $T_{\max} - \frac{\min\limits_{i\in K} p_i}{s_j}$.

Проверка выполнения этих ограничений позволяет добиться существенной экономии времени выполнения точного алгоритма, так как значительная часть распределений работ по процессорам отбраковывалась именно на этом этапе, и применять точный однопроцессорный алгоритм из разд. 1.4.1 приходилось в среднем на порядки реже, чем изначально предусмотренные m^{n+1} раз.

Как точный, так и эвристический алгоритмы были заданы в среде MatLab и запускались на персональном компьютере с процессором Pentium IV 3.2 Mhz. Все нижеизложенные результаты практических испытаний были получены именно таким образом. Для начала приведем сравнительную таблицу, в которой указано среднее время работы алгоритмов для различных размерностей задачи (число работ и число процессоров), усредненное по всем проведенным испытаниям для данной размерности.

Таблица 7. Среднее время работы точного и эвристического алгоритмов в зависимости от размерности задачи.

Размерность задачи		Время выполнения	
Процессоры	Работы	Эвристика	Точный
2	20	0.01 с	2.28 с.
3	25	0.06 с	457 с.
4	30	0.09 с	~13 ч.
6	40	0.14 с	>24 ч.
8	50	0.26 с	>24 ч.
8	100	0.69 с	>24 ч.
16	250	3.82 с	>24 ч.
16	500	10.23 с	>24 ч.
64	1000	257.63 с	>24 ч.

Из приведенной табл. 7 видно, что даже при производимой оптимизации точный алгоритм существенно менее эффективен, чем эвристический, и область его применимости ограничена задачами достаточно малой размерности. Уже в случае 6 процессоров и 40 работ точный алгоритм проработал более 24 часов, но так и не закончил свою работу, поэтому из практических соображений пришлось его прервать. Поэтому в этом случае и случаях еще большей размерности применять точный алгоритм не представлялось возможным, и для дальнейших сравнительных испытаний точного и эвристического алгоритмов мы ограничились случаем 3 процессоров и 25 работ как позволяющим за разумное время провести достаточно большое количество испытаний для набора репрезентативной статистики.

Для проверки сформулированной ранее гипотезы о том, что эвристический алгоритм работает тем корректнее, чем меньше время выполнения работ и загрузки данных процессорами по сравнению с $T_{\max}$, были введен следующий коэффициент, характеризующий интересующее нас соотношение между параметрами задачи:

$$\beta = \frac{\max\limits_{\substack{i=1,\dots,n \\ j=1,\dots,m}} \left(\frac{p_i}{s_j} + \frac{v_i}{l_j} \right)}{2T_{\max}} \qquad (7)$$

В числителе этого коэффициента стоит максимально возможное при заданных параметрах задачи время выполнения работы на процессоре $\max\limits_{\substack{i=1,\dots,n\\ j=1,\dots,m}} \frac{p_i}{s_j}$, а также максимально возможное время загрузки данных работы в память процессора $\max\limits_{\substack{i=1,\dots,n\\ j=1,\dots,m}} \frac{v_i}{l_j}$. Так как в числителе (7) находится сумма этих максимальных значений, а процессы загрузки данных и выполнения работ протекают параллельно на одном и том же временном интервале $(0;T_{\max}]$, то для более адекватного отображения отношения этих величин к длине общего директивного интервала, в знаменателе она удваивается.

Проведенные согласно указанному алгоритму 9 серий по 100 испытаний (для значений β в диапазоне от 0.2 до 1 с шагом 0.1) показали следующие результаты, приведенные ниже в Таблице 8.

Таблица 8. Результаты эксперимента: процент некорректной работы эвристического алгоритма в зависимости от отношения между длиной директивного интервала и максимальными длительностями выполнения работ и загрузки данных.

Параметр β	% некорректной работы
0.2	3
0.3	6
0.4	11
0.5	15
0.6	18
0.7	19
0.8	23
0.9	31
1.0	29

Таким образом, выдвинутая гипотеза получила экспериментальное подтверждение, и предложенный эвристический алгоритм может

быть рекомендован к практическому применению для решения сформулированной задачи о поиске допустимого расписания в случае, когда максимальные длительности выполнения работ и загрузки данных существенно меньше, чем длина директивного интервала всех работ.

1.5. Ограничения по памяти. Произвольный граф связей. Переключения с затратами. Время - дискретные такты

В предыдущем разделе был рассмотрен достаточно сложный вариант исследуемой задачи построения допустимого расписания в многопроцессорной системе жесткого реального времени, в котором из-за ограничений по памяти процессоров и временных затрат, требуемых на загрузку данных, одновременно с расписанием выполнения работ предложенные алгоритмы строили расписание загрузки данных в оперативную память процессоров. Тем не менее, для того, чтобы иметь возможность построить и применить действительно эффективные алгоритмы пришлось пойти на одно значительное упрощения исходной, сформулированной в начале главы задачи: директивные интервалы всех работ совпадали. При этом в многопроцессорном случае удалось построить только эвристический алгоритм, никак не использующий возможность переключения работ с процессора на процессор, в силу чего ни временные затраты на прерывания, ни граф связей между процессорами в алгоритмах, применяемых в предыдущем разделе были не важны (в первом точном алгоритме – в силу его «однопроцессорности», во втором – по построению). В настоящем разделе будет рассмотрен в какой-то мере еще более сложный вариант исходной задачи о поиске допустимого расписания, в котором связи между процессорами могут образовывать неполный граф, и единственным упрощающим допущением является то, что все процессоры работают дискретными тактами, синхронизованными во времени ([40], [41]).

Отметим, что это не те процессорные такты, которыми осуществляется работа всех реально существующие процессоров, а нечто гораздо более протяженное во времени, за которое процессор успевает совершить существенно больше, чем одну элементарную операцию. При этом также отметим, что, переходя к предельно малой протяжен-

ности такта, можно перейти к изначально сформулированному в начале данной работы непрерывному случаю.

Дадим точную постановку рассматриваемого в данной главе варианта исходной задачи ([44], [46]).

1.5.1. Постановка задачи

Рассматривается вычислительная система, состоящая из m процессоров, работа которых рассматривается как набор последовательных тактов. Такты всех процессоров синхронизованы во времени, т. е. их начала и концы для всех процессоров совпадают. Процессор j (j = 1,..., m) характеризуется производительностью s_j и объемом памяти V_j. Имеется n независимых работ, каждая работа i (i =1,..., n) характеризуется объемом p_i, директивным интервалом исполнения $(r_i, d_i]$, и объемом памяти v_i, который необходимо загрузить в процессор для ее выполнения.

Без ограничения общности можно считать, что все r_i, d_i – натуральные числа (т.е. начала и концы всех директивных интервалов задают номера соответствующих тактов работы процессоров) и $\min_i r_i = 0$, $\max_i d_i = T$ (т.е. в задаче рассматривается T тактов работы процессоров). Таким образом, s_j – это объем работы, выполняемой процессором j за один такт, т.е. для того, чтобы полностью выполнить работу i, процессору j необходимо затратить $\lceil p_i/s_j \rceil$ тактов. В фиксированный момент времени каждая работа может выполняться не более чем одним процессором, и каждый процессор может выполнять не более одной работы. Предполагается, что прерывание выполнения работы на процессоре j_1 с целью ее переключения на одном из последующих тактов на процессор j_2 или для возобновления на этом же процессоре ($j_2 = j_1$) требует дополнительной работы процессоров j_1 и j_2 в суммарном объеме $\lambda_{j_1 j_2}$.

Связи между процессорами могут меняться во времени, что определяется массивом I размером $m \times T \times m \times T$, при этом $I(j_1, k_1, j_2, k_2) = 1$ если при выполнении работы на процессоре j_1 в конце такта k_1 ее можно прервать и возобновить на процессоре j_2 в начале

такта k_2, и $I(j_1,k_1,j_2,k_2)=0$ если такое переключение невозможно $\left(1\le j_1, j_2 \le m;\ 1\le k_1 < k_2 \le T\right)$.

Необходимо построить расписание выполнения работ на процессорах, при котором все работы выполняются в своих директивных интервалах, или показать, что такого расписания не существует.

1.5.2. Построение сети

Для решения сформулированной выше задачи построим многопродуктовую потоковую сеть $N = (V, E)$, где V – вершины, E – дуги (рис. 11).

Множество вершин сети $V(N)$ состоит из:

1) n источников $u_1, u_2,\ldots,u_n$ по количеству задач.

2) n стоков $w_1, w_2,\ldots,w_n$ по количеству задач.

3) T ярусов (каждый по m пар вершин) соответствующих рассматриваемым в задаче тактов. Обозначим эти пары вершин как $\left(x_j^k, y_j^k\right)$, $j=1,..,\ m;\ \ k=1,..,\ T$. Пронумеруем эти ярусы от 1 до T.

Нетрудно заметить, что $\left|V\right| = 2n + 2mT$.

Перейдем теперь к дугам сети. Множество дуг $E(N)$ состоит из:

1) Дуг, соединяющих пары $\left(x_j^k, y_j^k\right)$ в каждом из T ярусов.

2) Дуг $\left(u_i, x_j^k\right)$, соединяющих, для каждого i источник u_i со всеми вершинами x_j^k, $j=1,...,\ m$, $k \in (r_i, d_i)$.

3) Дуг $\left(y_j^k, w_i\right)$, соединяющих для каждого i сток w_i со всеми вершинами y_j^k, $j=1,...,\ m\ \ k \in (r_i, d_i)$.

4) Дуг $\left(y_{j_1}^{k_1}, x_{j_2}^{k_2}\right)$, для каждого i соединяющих все возможные $y_{j_1}^{k_1}$ с $x_{j_2}^{k_2}$, такие, что $k_1 < k_2$ и $I(j_1,k_1,j_2,k_2)=1$, $k_1, k_2 \in \left(r_i, d_i\right)$, $j_1, j_2 = 1,..., m$.

Полученная в результате этого построения сеть $N = (V, E)$ приведена на рис. 11. Таким образом, наличие в сети N дуг $\left(u_i, x_j^k\right)$ и $\left(u_i, x_j^{k+1}\right)$ означает, что работу i можно начать выполнять на процессоре j как на k-м, так и на $(k+1)$-м такте, а наличие дуг $\left(y_j^k, w_i\right)$ и

$\left(y_j^{k+1}, w_i\right)$ означает, что завершить работу i можно как в конце k-го, так и в конце (k+1)-го такта. Наличие дуги $\left(y_{j_1}^k, x_{j_2}^{k+1}\right)$ означает, что выполнение работы на процессоре j_1 в конце k-го такта можно прервать и возобновить на процессоре j_2 в начале (k+1)-го такта.

Подсчитаем количество дуг. Количество дуг 1-го типа – mT. Количество дуг второго типа можно оценить сверху для каждой работы как mT (случай, когда директивный интервал представляет собой все рассматриваемые T тактов). Так как всего работ n, получаем для количества дуг этого типа оценку, равную nmT. Для дуг третьего типа рассуждения полностью аналогичны второму, поэтому для этого типа дуг также получаем nmT. Для оценки количества ребер четвертого типа, рассмотрим все ребра первого типа как вершины, соединяющиеся между собой. Получаем всего mT вершин. Заметим, что каждая такая вершина в предельном случае, когда существует по крайней мере одна работа с директивным интервалом (0, T) может соединяться со всеми себе подобными, кроме вершин принадлежащих своему ярусу сети (т.к. $k_1 < k_2$). Всего таких соединений получается $m(T-1)$. То есть, всего $m^2T(T-1)$ соединений. Так как каждую дугу мы при таком подсчете посчитали два раза, то полученный результат нужно еще разделить пополам, и тогда получим $\frac{1}{2}m^2T(T-1)$ дуг 4-го типа.

Таким образом, общее количество дуг в построенной сети не превышает $\frac{1}{2}m^2T(T-1)+2nmT+mT=mT\left(\frac{1}{2}m(T-1)+2n+1\right)$, т.е. $|E|\le mT\left(\frac{1}{2}m(T-1)+2n+1\right)$.

Каждая дуга сети N имеет два параметра: пропускную способность и стоимость единицы потока по дуге. Пропускные способности всех дуг полагаем равными 1. Стоимость прохождения единицы потока по дуге $\left(x_j^k, y_j^k\right)$ полагаем равной s_j, по дуге $\left(y_{j_1}^{k_1}, x_{j_2}^{k_2}\right)$ - равной $-\lambda_{j_1 j_2}$, а по дугам $\left(u_i, x_j^k\right)$ и $\left(y_j^k, w_i\right)$ - равной 0.

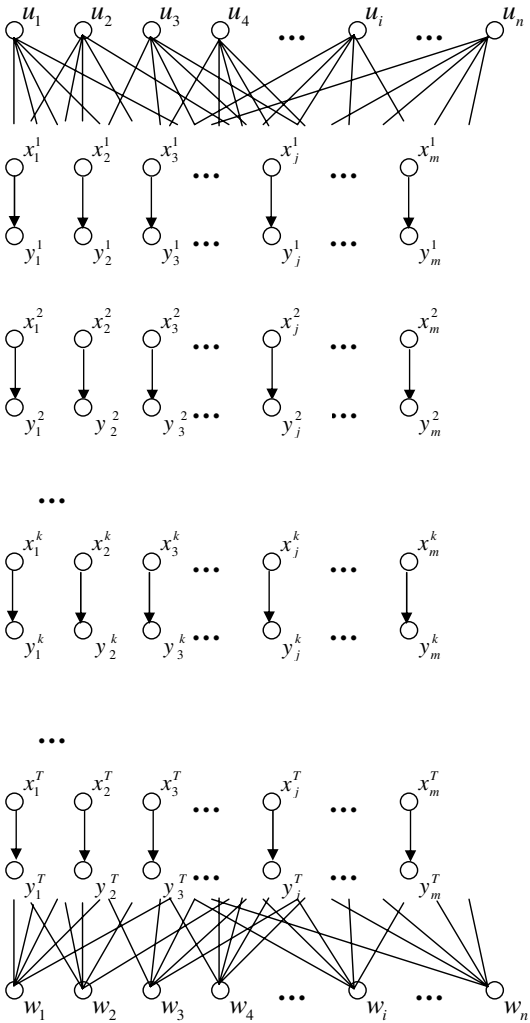

u_1 u_2 u_3 u_4 ... u_i ... u_n
x_1^1 x_2^1 x_3^1 ... x_j^1 ... x_m^1
y_1^1 y_2^1 y_3^1 ... y_j^1 ... y_m^1
x_1^2 x_2^2 x_3^2 ... x_j^2 ... x_m^2
y_1^2 y_2^2 y_3^2 ... y_j^2 ... y_m^2
...
x_1^k x_2^k x_3^k ... x_j^k ... x_m^k
y_1^k y_2^k y_3^k ... y_j^k ... y_m^k
...
x_1^T x_2^T x_3^T ... x_j^T ... x_m^T
y_1^T y_2^T y_3^T ... y_j^T ... y_m^T
w_1 w_2 w_3 w_4 ... w_i ... w_n

Рис. 11. Сеть N.

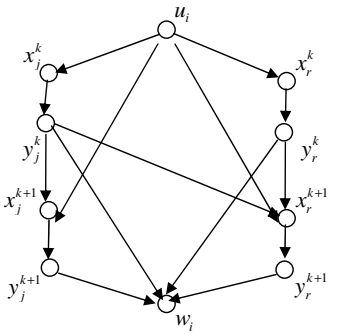


Рис. 12. Фрагмент сети N.

Рассмотрим в этой сети многопродуктовый поток, состоящий из n типов потоков, причем поток i-го продукта исходит из источника u_i и входит в сток w_i ($i = 1,...,n$). Эти потоки описывают выполнение работ многопроцессорной системой. Будем интерпретировать поток i-го продукта по дугам сети N следующим образом:

1) по дуге $\left(u_i, x_j^k\right)$ - начало выполнения работы i (на такте k);

2) по дуге $\left(x_j^k, y_j^k\right)$ - выполнение работы i процессором j на такте k;

3) по дуге $\left(y_{j_1}^{k_1}, x_{j_2}^{k_2}\right)$ - прерывание выполнения работы i процессором j_1 на такте k_1 и ее переключение на процессор j_2 на такте k_2;

4) по дуге $\left(y_j^k, w_i\right)$ - конец выполнения работы i (на такте k).

При прохождении потока i-го продукта по дуге $\left(x_j^k, y_j^k\right)$ к его стоимости прибавляется величина, равная s_j (объем выполненной работы на j-м процессоре за один такт). При прохождении потока по дуге $\left(y_{j_1}^{k_1}, x_{j_2}^{k_2}\right)$, которая соответствует прерыванию или переключению работы, из его стоимости вычитается величина λ_{j_1} (где j_1 – номер процессора, на котором прервали работу). При прохождении потока по дугам $\left(u_i, x_j^k\right)$ и $\left(y_j^k, w_i\right)$ его стоимость остается без изменений. Бу-

дем обозначать поток *i*-го продукта по дуге (*a, b*) как $f^i(a,b)$. Заметим, что в нашей модели $f^i(a,b) \in \{0,1\}$ при всех *a*, *b*, и *i*.

Например, если для сети, изображенной на рис. 12, $f^i\left(x_{j_1}^k, y_{j_1}^k\right) = f^i\left(y_{j_1}^k, x_{j_2}^{k+1}\right) = f^i\left(x_{j_2}^{k+1}, y_{j_2}^{k+1}\right) = 1$, то на *k*-м и (*k*+1)-м тактах суммарный объем работы процессоров j_1 и j_2 по выполнению работы *i* составляет $s_{j_1} + s_{j_2} - \lambda_{j_1 j_2}$.

Лемма 7. Сформулированная в разд. 1.5 задача является NP-трудной.

1.5.3. Необходимые и достаточные условия существования допустимого расписания

Докажем центральное утверждение данного раздела, позволяющее сводить исходную задачу определения существования и построения допустимого расписания к задаче поиска многопродуктового потока, обладающего рядом свойств, в построенной в разд. 1.5.2 сети *N* ([45], [47]).

Теорема 2. Допустимое расписание существует тогда и только тогда, когда в сети *N* существует *n*-продуктовый поток, обладающий следующими свойствами:

$$\sum_{j,k} f^i\left(x_j^k, y_j^k\right) s_j - \sum_{j_1, j_2, k_1, k_2 : k_1 < k_2} f^i\left(y_{j_1}^{k_1}, x_{j_2}^{k_2}\right) \lambda_{j_1 j_2} \geq p_i \qquad (8)$$

при всех $i = 1,..., n$, т.е. суммарная стоимость потока *i*-го продукта составляет не менее p_i;

$$\sum_{i, r_1, r_2 : r_1 < k < r_2} f^i\left(y_j^{r_1}, x_j^{r_2}\right) v_i + \sum_{i=1}^{n} f^i\left(x_j^k, y_j^k\right) v_i \leq V_j \qquad (9)$$

при всех $j = 1,\ldots, m$; $k = 1,\ldots, T$, т.е. на каждом такте выполнено ограничение по объему памяти каждого процессора.

В качестве иллюстрации рассмотрим простой пример, в котором значения параметров условий задачи равны следующим величинам: $T = 5$; $n = 3$; $m = 2$; $s_1 = s_2 = 1$; $p_1 = p_2 = p_3 = 3$; $r_1 = r_2 = r_3 = 0$; $d_1 = d_2 = d_3 = 5$; $\lambda_{12} = \lambda_{21} = 1$; $I(j_1, k_1, j_2, k_2) = 1$ при всех $1 \leq j_1, j_2 \leq 2$; $1 \leq k_1 < k_2 \leq 5$; $v_1 = v_2 = v_3 = 1$; $V_1 = V_2 = 1$. На рис. 13 изображен фрагмент сети *N* для

данного примера, включающий только те дуги, поток по которым равен 1.

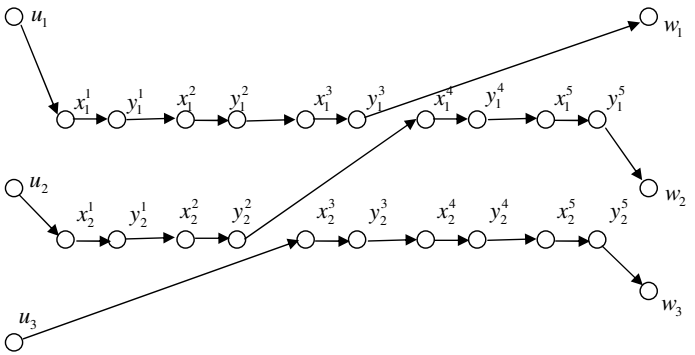


Рис. 13. Потоки в сети N, отображающие допустимое расписание для рассматриваемого примера.

Поток первого продукта проходит по пути $u_1 - x_1^1 - y_1^1 - x_1^2 - y_1^2 - x_1^3 - y_1^3 - w_1$ и его стоимость равна 3, т.е. первая работа выполняется первым процессором на первых трех тактах и получает три единицы процессорного времени. Поток второго продукта проходит по пути $u_2 - x_2^1 - y_2^1 - x_2^2 - y_2^2 - x_1^4 - y_1^4 - x_1^5 - y_1^5 - w_2$ и его стоимость равна $2-1+2=3$, т.е. вторая работа выполняется первые два такта вторым процессором, затем переключается на первый процессор, на котором выполняется четвертый и пятый такты. Суммарный объем работы рассматриваемой двухпроцессорной системы по выполнению второй работы равен 4 единицам процессорного времени, однако одна из этих единиц расходуется на переключение, и на выполнение работы остается необходимые 3 единицы. Поток третьего продукта проходит по пути $u_3 - x_2^3 - y_2^3 - x_2^4 - y_2^4 - x_2^5 - y_2^5 - w_3$ и его стоимость равна 3, т.е. третья работа выполняется вторым процессором на третьем, четвертом и пятом тактах и также получает три единицы процессорного времени. Таким образом, условие (8) Теоремы 1 выполнено. Нетрудно заметить, что первое слагаемое в левой части условия (9) Теоремы 1 всегда равно 0, а второе равно 1, поэтому усло-

вие (9) также выполняется. Таким образом, допустимое расписание существует и определяется матрицей $A = \begin{pmatrix} 11122 \\ 22333 \end{pmatrix}$.

1.5.4. Алгоритм построения расписания

Итак, мы показали, что предлагаемое сведение задачи о поиске многопроцессорного расписания к задаче о поиске многопродуктового потока в сети N корректно и однозначно интерпретировали одну задачу в терминах другой. Поскольку поток по каждой дуге равен 0 или 1, то для нахождения искомого потока можно воспользоваться методами булевого линейного программирования ([15, 16]). Лучший из подобных алгоритмов поиска основан на быстром алгоритме перемножения матриц и требует $O\left(l^{3.5} r^3 q^5 \log(rDU)\right)$, где l – количество потоков, r – количество узлов сети, q – количество ее дуг, D – максимальная требуемая суммарная стоимость потока, U – максимальная пропускная способность дуг сети ([16]). В рассматриваемом случае $l = n$, $r = 2n + 2mT$, $q = mT\left(\frac{1}{2}m(T-1) + 2n + 1\right)$, $D = \max\limits_i p_i$, $U = 1$. Таким образом, в терминах условий исходной задачи вычислительная сложность предложенного алгоритма составляет $O\left(\left(n^{6.5} m^5 T^5 + n^{3.5} m^8 T^8\right)\left(m^5 T^5 + n^5\right)\log\left(2m(T+1)\max\limits_i p_i\right)\right)$.

Таким образом, мы построили псевдополиномиальный алгоритм для решения сформулированной в данном разделе NP-трудной задачи о поиске допустимого расписания в многопроцессорной системе реального веремни с ограничениями по памяти процессоров, с произвольным графом связей между процессорами и возможностью прерывания и переключения работ, сопряженным с дополнительными временными затратами. Однако, несмотря на псевдополиномиальность полученного алгоритма, его практическое применение в многопроцессорных системах даже не очень крупного масштаба затруднено вследствие высоких степеней полинома. Поэтому в данном случае может иметь практический смысл также и применение различных эвристических алгоритмов типа EDF, рассмотренных в разд. 1.3.

1.6. Задача синтеза

В предыдущих разделах мы рассмотрели задачу построения допустимого расписания в системе жесткого реального времени в большом количестве ее вариаций и для каждой предложили методы ее решения. В этой же завершающем настоящую главу разделе предлагается рассмотреть своего рода обратную задачу к рассматриваемой до сих пор, а именно, задачу построения такой системы жесткого реального времени, чтобы допустимое расписание заведомо существовало при заданных условиях задачи.

1.6.1. Отсутствие ограничений на память процессоров

В [18] была рассмотрена задача синтеза в случае отсутствия ограничений по памяти процессоров и периодически поступающих работах. Интерпретируем ее в случае апериодических директивных интервалов и приведем ее основной результат.

1.6.1.1. Постановка задачи

Имеется n работ, каждая из которых определяется своим директивным сроком начала r_i и директивным сроком окончания d_i, т.е. директивным интервалом $A_i = (r_i, d_i]$, $i = 1,...,n$, а также сложностью p_i, $i = 1,...,n$. Имеется m процессоров различной вычислительной мощности s_j, $j = 1,\ldots,m$. Работа сложности p_i может быть выполнена на процессоре вычислительной мощности s_j за время $t_{ij} = p_i / s_j$. В фиксированный момент времени каждая работа может выполняться не более чем одним процессором, и каждый процессор может выполнять не более одной работы. При выполнении работ допускаются прерывания и переключения с одного процессора на другой. Затраты процессоров на прерывания пренебрежимо малы.

Требуется определить множество векторов $(s_1,...,s_m)$ производительностей и объемов памяти процессоров, для которых задача построения допустимого расписания имеет решение.

1.6.1.2. Построение области допустимых параметров процессоров

Пусть $\tau_0 < \tau_1 < ... < \tau_k$ - все различные значения r_i и d_i, i = 1,…,n, $I_l = (\tau_{l-1}; \tau_l]$, l = 1,…, h. Пусть δ_l - длина интервала I_l. По условиям задачи, работа i доступна для выполнения (*выполнима*) в момент времени t, если $t \in A_i$. Заметим, что внутри каждого интервала I_l набор выполнимых работ остается постоянным. Поэтому будем говорить, что работа i доступна в интервале I_l, если $I_l \subseteq A_i$. Кроме того, директивный интервал каждой работы $A_i = (r_i, d_i]$ является объединением некоторых I_l. Среди интервалов I_l в дальнейшем будем рассматривать только те, которые принадлежат некоторому директивному интервалу A_i. Без ограничения общности можно считать, что все I_l (l = 1,…, h) удовлетворяют этому требованию.

Обозначим множество индексов работ как N, т.е. N = {1,…, n}. Как следует из [4], допустимое расписание в рассматриваемой системе существует тогда и только тогда, когда для любого подмножества $N_j \subseteq N$ выполнено неравенство:

$$\sum_{l=1}^{h} \delta_l S_{k(l,N_j)} \geq \sum_{i \in N_j} p_i \ , \qquad (10)$$

где $k(l, N_j) = \min(m, m')$, m' – число работ $i \in N_j$, доступных в интервале I_l, $S_k = \sum_{i=1}^{k} s_i$, $s_1 \geq s_2 \geq ... \geq s_m$.

Пусть $D(i)$ ($i \in N$) – множество всех интервалов I_l, в которых работа i доступна. Для всех $1 \leq l_1 \leq l_2 \leq h$ определим следующие множества:

$$\tilde{I}^1_{l_1 l_2} = \{I_{l_1}, ..., I_{l_2}\}, \ \tilde{I}^2_{l_1 l_2} = \{I_1, ..., I_{l_1-1}, I_{l_2+1}, ..., I_h\},$$

$$\tilde{N}^1_{l_1 l_2} = \{i \in N : D(i) \subseteq \tilde{I}^1_{l_1 l_2}\}, \ \tilde{N}^2_{l_1 l_2} = \{i \in N : D(i) \subseteq \tilde{I}^2_{l_1 l_2}\},$$

$$\bar{N}^1_{l_1 l_2} = \begin{cases} \tilde{N}^1_{l_1 l_2}, \text{ если } \bigcup\limits_{i \in \tilde{N}^1_{l_1 l_2}} D(i) = \tilde{I}^1_{l_1 l_2} \\ \varnothing \text{ в противном случае} \end{cases},$$

$$\bar{N}^2_{l_1 l_2} = \begin{cases} \tilde{N}^2_{l_1 l_2}, \text{ если } \bigcup\limits_{i \in \tilde{N}^2_{l_1 l_2}} D(i) = \tilde{I}^2_{l_1 l_2} \\ \varnothing \text{ в противном случае} \end{cases}.$$

С помощью рассуждений, аналогичных тем, которые приводятся в [1], можно показать, что допустимое расписание в рассматриваемой системе существует в том и только в том случае, когда неравенство (10) справедливо для всех l_1, l_2 $\left(1 \le l_1 \le l_2 \le h\right)$ и всех подмножеств $\bar{N}^1 \subseteq \bar{N}^1_{l_1 l_2}$, $\bar{N}^2 \subseteq \bar{N}^2_{l_1 l_2}$, для которых $\bigcup_{i \in \bar{N}^1} D(i) = \tilde{I}^1_{l_1 l_2}$, $\bigcup_{i \in \bar{N}^2} D(i) = \tilde{I}^2_{l_1 l_2}$. С учетом этого замечания будем исследовать неравенство (10) для подмножеств N_j, где j принадлежит некоторому множеству индексов J. Очевидно, что $|J| \le 2^n - 1$, а в случае, когда выполнено условие регулярности [1], $|J| \le h(h+1)$. Перепишем неравенство (10) в виде:

$$k_1^j S_1 + k_2^j S_2 + \ldots + k_m^j S_m \ge Q^j, j \in J, \qquad (11)$$

где $k_i^j = \sum_{l: k(l, N_j) = i} \delta_l$ $(i = 1, \ldots, m)$; $Q^j = \sum_{i \in N_j} p_i$.

Отметим, что

$$S_1 \le S_2 \le \ldots \le S_m.$$

Лемма 7. Неравенство (11) справедливо тогда и только тогда, когда

$$S_m \ge \max_{j \in J} \frac{Q^j}{k_1^j + \ldots + k_m^j},$$

$$S_r \ge \max_{j \in J_r} \frac{Q^j - k_m^j S_m - \ldots - k_{r+1}^j S_{r+1}}{k_1^j + \ldots + k_r^j},$$

где $J_r = \left\{ j \in J, k_1^j + \ldots + k_r^j \ne 0 \right\}$, $r = 1, \ldots, m$ -1.

Лемма 8. Пусть

$$S_k = s_1 + s_2 + \ldots + s_k \ \ (k = 1,2,\ldots, m).$$

Для того, чтобы по заданным величинам S_1, …, S_m можно было определить величины s_1, …, s_m, такие, что

$$s_1 \ge s_2 \ge \ldots \ge s_m \ge 0$$

необходимо и достаточно выполнение неравенств:

$$0 \le S_1 \le S_2 \le \ldots \le S_m,$$

$$2S_1 \ge S_2,$$

$$2S_r \ge S_{r-1} + S_{r+1} \ (r = 2, \ldots, m\text{-}1).$$

При этом

$$s_1 = S_1,\ s_r = S_r - S_{r\text{-}1} \quad (r = 2,\ldots,\ m). \tag{12}$$

Таким образом, из утверждений лемм 7 и 8 следует, что для определения допустимых производительностей процессоров необходимо и достаточно определить величины $S_1,\ldots,\ S_m$, которые задаются следующими неравенствами:

$$S_m \ge \max_{j \in J} \frac{Q^j}{k_1^j + \ldots + k_m^j} \tag{13}$$

$$S_{r+1} \ge S_r \ge \max\left(\max_{j \in J_r} \frac{Q^j - k_m^j s_m - \ldots - k_{r+1}^j s_{r+1}}{k_1^j + \ldots + k_r^j}, \frac{S_{r+1} + S_{r-1}}{2} \right), \tag{14}$$

$$r = 2,\ \ldots,\ m\text{-}1$$

$$S_2 \ge S_1 \ge \max\left(\max_{j \in J_1} \frac{Q^j - k_m^j s_m - \ldots - k_2^j s_2}{k_1^j}, \frac{1}{2} S_2 \right) \tag{15}$$

Для поиска допустимых решений нужно решить систему неравенств (13)-(15) и найти вектор $(S_1,\ldots,S_m)$, а затем согласно (12) определить искомые величины $s_1,\ldots,s_m$. Решение же различных оптимизационных задач (например, минимизация суммы производительностей процессоров) в данном случае сводится к решению задачи целочисленного линейного программирования.

1.6.2. Учет ограничений на память процессоров

В разд. 1.6.1 мы рассмотрели задачу синтеза системы реального времени и указали систему неравенств, которой должны удовлетворять параметры системы ($s_1,\ldots,s_m$) для существования допустимого расписания при заданном множестве работ. Далее мы построим сис-

тему аналогичных ограничений для многопроцессорной системы реального времени с учетом ограничений на память процессоров. Сформулируем эту задачу более строго.

1.6.2.1. Постановка задачи

Имеется n работ, каждая из которых определяется своим директивным сроком начала r_i и директивным сроком окончания d_i, т.е. директивным интервалом $A_i = (r_i, d_i]$, $i = 1,...,n$, а также сложностью p_i, $i = 1,...,n$ и объемом данных, которые необходимо загрузить в память процессора для ее выполнения v_i, $i = 1,...,n$. Имеется m процессоров различной вычислительной мощности s_j, $j = 1,\ldots,m$ и различного объема памяти V_j, $j = 1,\ldots,m$. Работа сложности p_i может быть выполнена на процессоре вычислительной мощности s_j за время $t_{ij} = p_i / s_j$. В фиксированный момент времени каждая работа может выполняться не более чем одним процессором, и каждый процессор может выполнять не более одной работы. Для выполнения работы (или какой-либо ее части) на каком-либо процессоре все ее данные должны быть загружены в память этого процессора в полном объеме. При выполнении работ допускаются прерывания и переключения с одного процессора на другой. Затраты процессоров на прерывания, а также на загрузку и выгрузку работ из памяти процессоров пренебрежимо малы. Без ограничения общности можно считать, что $v_1 \geq ... \geq v_n$ и $s_1 \geq ... \geq s_m$.

Требуется определить множество векторов $(s_1,...,s_m,V_1,...,V_m)$ производительностей и объемов памяти процессоров, для которых задача построения допустимого расписания имеет решение.

1.6.2.2. Построение области допустимых параметров процессоров

Пусть $\tau_0 < \tau_1 < ... < \tau_k$ - все различные значения r_i и d_i, $i = 1,\ldots,n$, $I_l = (\tau_{l-1}; \tau_l]$, $l = 1,\ldots, h$. Пусть δ_l - длина интервала I_l. По условиям задачи, работа i доступна для выполнения (*выполнима*) в момент времени t, если $t \in A_i$. Заметим, что внутри каждого интервала I_l набор

выполнимых работ остается постоянным. Поэтому будем говорить, что работа *i* доступна в интервале I_l, если $I_l \subseteq A_i$. Кроме того, директивный интервал каждой работы $A_i = (r_i, d_i]$ является объединением некоторых I_l. Среди интервалов I_l в дальнейшем будем рассматривать только те, которые принадлежат некоторому директивному интервалу A_i. Без ограничения общности можно считать, что все I_l ($l = 1,\ldots, h$) удовлетворяют этому требованию.

Обозначим множество индексов работ как *N*, т.е. $N = \{1,\ldots, n\}$. Построим систему ограничений, которым должны удовлетворять величины $s_1,...,s_m$ и $V_1,...,V_m$ для того, чтобы допустимое расписание существовало при заданном множестве работ. Сначала приведем ряд дополнительных соображений касательно ограничений по памяти процессоров.

Лемма 9. В рассматриваемой постановке задачи синтеза ограничения по памяти процессоров эквивалентны невозможности выполнить некоторые работы на некоторых процессорах.

Исходя из утверждения леммы ограничения по объему памяти процессоров удобно закодировать матрицей возможности выполнения определенных работ на определенных процессорах *E* размера ($n \times m$), элемент которой $E_{ij} = 1$, если i-ю работу можно выполнять на j-м процессоре (то есть $V_j \geq v_i$), и $E_{ij} = 0$ в противном случае. Исходя из смысла задаваемых ею ограничений, матрица *E* не может быть произвольной: если какой-либо процессор *j* может выполнить некоторую работу *i* с объемом данных v_i, то это означает, что он может выполнить и все такие работы i', что $v_i \geq v_{i'}$. Поэтому если $E_{ij} = 1$, то обязательно $E_{kj} = 1,\ k = i+1,...,n$. Для существования допустимого расписания также необходимо, чтобы по крайней мере один процессор мог выполнять работу с максимальным объемом данных (в наших обозначениях это первая работа). Изложенные в этом абзаце соображения формально задаются следующими ограничениями:

$$E_{ij} - E_{i+1,j} \leq 0,\ i = 1,...,n-1;\ j = 1,...,m \qquad (16)$$

$$\sum_{j=1}^{m} E_{1j} \geq 1 \qquad (17)$$

Введем также матрицу E', каждый элемент которой получается из соответствующего элемента матрицы E следующим образом:

1) $E'_{1j} = E_{1j}$ для $j = 1,\ldots,m.$

2) Если $E_{ij} = 0$, то $E'_{ij} = 0$ для $i = 2,...,n$ и $j = 1,...,m$.

3) Если $E_{ij} = 1$ и $E_{i-1,j} = 0$, то $E'_{ij} = 0$ для $i = 2,...,n$ и $j = 1,...,m$.

4) Если $E_{ij} = 1$ и $E_{i-1,j} = 1$, то $E'_{ij} = 1$ для $i = 2,...,n$ и $j = 1,...,m$.

Таким образом, если рассмотреть матрицу разности $E - E'$, то в ней в каждом столбце, отвечающем за некоторый процессор, будет ровно одна единица – на той позиции, которая обозначает работу с максимально возможным объемом данных из тех, которые этот процессор заведомо может выполнить. Отсюда легко видеть, что при определенной таким образом матрице E граничные значения объемов памяти процессоров относительно объемов данных работ задаются векторным равенством

$$\mathbf{V}=\left(E - E'\right)^{\mathrm{T}} \mathbf{v} \qquad (18)$$

где **V**- вектор, образованный упорядоченными объемами памяти процессоров $V_1,...,V_m$, **v** – вектор, образованный упорядоченными объемами данных всех работ $v_1,...,v_n$.

В качестве иллюстрации к введенным обозначениям рассмотрим следующий простой пример: пусть $n = 4$, $v_1 = 7$, $v_2 = 5$, $v_3 = 3$, $v_4 = 2$. Имеется также 3 процессора. Пусть на первом можно выполнить вторую, третью и четвертую работы, на втором - любую из работ, а на третьем – только последнюю. Это будет означать, что

$$E = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}, \qquad E' = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}, \qquad E - E' = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Отсюда (33) запишется как:

$$\begin{pmatrix} V_1 \\ V_2 \\ V_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 7 \\ 5 \\ 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 5 \\ 7 \\ 2 \end{pmatrix},$$

то есть $V_1 = 5,\ V_2 = 7,\ V_3 = 2$.

В [3] приводятся необходимые и достаточные условия существования допустимого расписания в системе реального времени идентичной рассматриваемой, только без учета прерываний. Со ссылкой на результат из [19] там доказано, что допустимое расписание на каждом из интервалов I_l существует тогда и только тогда, когда выполнена следующая серия неравенств:

$$\begin{aligned} q_1 &\le s_1 \delta_l && \text{(1-е неравенство)} \\ q_1 + q_2 &\le (s_1 + s_2)\delta_l && \text{(2-е неравенство)} \\ &\vdots && (19) \\ q_1 + q_2 + \ldots + q_{m-1} &\le (s_1 + \ldots + s_{m-1})\delta_l && \text{((}m\text{-1)-е неравенство)} \\ q_1 + q_2 + \ldots + q_r &\le (s_1 + \ldots + s_m)\delta_l && (m\text{-е неравенство}) \end{aligned}$$

где $q_1 \ge \ldots \ge q_r$ - части работ ($r \le n$), назначеные неким существующим допустимым расписанием на выполнение на интервале I_l. При введении ограничений по памяти (как уже было сказано выше, эти ограничения эквивалентны ограничениям на выполнение определенных работ на определенных процессорах с дополнительными условиями (16) - (18)), указанная выше система неравенств (19) должна быть дополнена серией ограничений, характеризующей то, на каких процессорах может быть выполнена каждая из работ. Обозначим как $u_1 \ge \ldots \ge u_r$ объемы данных, необходимые для выполнения каждой из *r* работ, назначенных допустимым расписанием на интервал I_l, и перенумеруем все ранее введенные q_i в соответствии с этим порядком. Обозначим как F^l матрицу размера $r \times m$, составленную из всех строк матрицы *E*, соотвествующим работам из множества доступных на I_l. Выпишем эти неравенства на основании того, на каком множестве процессоров

может быть выполнена каждая работа и все работы, объем данных которых меньше данной:

$$\left(\sum_{j=1}^{m} F_{1j}^{l} s_j\right)\delta_l \geq q_1 \qquad \text{(1-е неравенство)}$$

$$\left(\sum_{j=1}^{m} F_{2j}^{l} s_j\right)\delta_l \geq q_1 + q_2 \qquad \text{(2-е неравенство)}$$

$$\vdots \qquad (20)$$

$$\left(\sum_{j=1}^{m} F_{rj}^{l} s_j\right)\delta_l \geq q_1 + \ldots + q_r \qquad (r\text{-е неравенство})$$

Для того, чтобы допустимое расписание для заданного множества N работ существовало в какой-либо системе реального времени необходимо и достаточно, чтобы оно существовало для любого подмножества $N_j \subseteq N$ в этой же системе. Поэтому, произвольно выбрав какое-либо подмножество N_j, можно записать для него на каждом из интервалов I_l системы неравенств (19) и (20), а затем просуммировать последние неравенства этих двух систем по всем интервалам I_l, $l = 1,...,h$. В правой части в обоих этих случаях будет сумма всех кусков всех работ из множества N_j по всем интервалам, то есть $\sum_{i \in N_j} p_i$.

Сами же неравенства (19) и (20) при подобном суммировании преобразуются соответственно в:

$$\sum_{l=1}^{h}\left(\delta_l \sum_{i=1}^{k(l,N_j)} s_i\right) \geq \sum_{i \in N_j} p_i \qquad (21)$$

$$\sum_{l=1}^{h}\left(\delta_l \sum_{i=1}^{k(l,N_j)} E_{g(N_j),i} s_i\right) \geq \sum_{i \in N_j} p_i \qquad (22)$$

Здесь $k\left(l,N_j\right) = \min\left(m,m'\right)$, где m'-число работ $i \in N_j$, доступных в интервале I_l. $g\left(N_j\right) = \arg\min_{i \in N_j} v_i$, то есть номер работы с минимальным объемом данных, если рассматривать только работы из множества N_j. Следует обратить внимание на то, что номер здесь берется от-

носительно всех n работ, а объемы данных работ были ранее упорядочены как $v_1 \geq ... \geq v_n$.

Теорема 3. Для того, чтобы в многопроцессорной системе реального времени, сформулированной в п.1.6.2.1, существовало допустимое расписание, необходимо и достаточно, чтобы была выполнена следующая система неравенств:

$$\sum_{l=1}^{h}\left(\delta_l \sum_{i=1}^{k(l,N_j)} E_{g(N_j),i} s_i\right) \geq \sum_{i\in N_j} p_i \qquad \text{(для любого } N_j \subseteq N\text{)} \tag{23}$$

$$E_{ij} - E_{i+1,j} \leq 0,\ i = 1,...,n-1;\ j = 1,...,m \tag{24}$$

$$\sum_{j=1}^{m} E_{1j} \geq 1 \tag{25}$$

$$\mathbf{V} \geq \left(E - E'\right)^{\mathbf{T}} \mathbf{v} \tag{26}$$

где $\mathbf{V}$- вектор, образованный упорядоченными объемами памяти процессоров $V_1,...,V_m$, $\mathbf{v}$ – вектор, образованный упорядоченными объемами данных всех работ $v_1,...,v_n$; I_l - интервалы образованные всеми различными началами и концами директивных интервалов работ, h – количество этих интервалов, δ_l – их длины; $k\left(l,N_j\right) = \min\left(m,m'\right)$, где m'-число работ $i \in N_j$, доступных в интервале I_l. $g\left(N_j\right) = \arg\min_{i\in N_j} v_i$, то есть номер работы с минимальным объемом данных, если рассматривать только работы из множества N_j. Матрица E имеет размер $n \times m$, ее элементы $E_{ij} \in \{0,1\}$. Размеры матрицы E' совпадают с размерами матрицы E, а каждый ее элемент получается из соответствующего элемента матрицы E следующим образом: $E'_{1j} = E_{1j}$ для $j = 1,\ldots,m$; если $E_{ij} = 0$, то $E'_{ij} = 0$ для $i = 2,...,n$ и $j = 1,...,m$; если $E_{ij} = 1$ и $E_{i-1,j} = 0$, то $E'_{ij} = 0$ для $i = 2,...,n$ и $j = 1,...,m$; если $E_{ij} = 1$ и $E_{i-1,j} = 1$, то $E'_{ij} = 1$ для $i = 2,...,n$ и $j = 1,...,m$.

Отсюда для поиска допустимых решений нужно решить систему неравенств (23)-(26) относительно производительностей процессоров

$(s_1,...,s_m)$ и всех элементов матрицы E, а затем согласно (26) и определения матрицы E' определить допустимые объемы памяти процессоров $V_1,...,V_m$. Решение же различных оптимизационных задач (например, минимизация суммы производительностей процессоров и суммарного объема памяти процессоров) в данном случае сводится к решению задачи целочисленного квадратичного программирования (ЦКП), так как в отличие от задачи синтеза без ограничений по памяти, здесь в серии неравенств (23) встречаются произведения переменных вида $E_{ij}s_k$. При этом заметим, что большая часть переменных в построенной задаче ЦКП – это булевские переменные (все E_{ij}). Относительно эффективные подходы решения именно такого класса задач ЦКП, основанные на методе ветвей и границ изложены, например, в [20] и [21].

Глава 2.
Разработка алгоритмов составления многопроцессорных расписаний без прерываний

В настоящей главе рассматриваются задачи составления многопроцессорных расписаний без прерываний. Исторически появлению дискретных методов оптимизации предшествовало развитие методов линейного программирования. Высокая вычислительная производительность этих методов позволила предполагать, что на их основе могут быть разработаны эффективные методы решения дискретных задач оптимизации. Однако вычислительные эксперименты не подтвердили это предположение. Принципиальные трудности решения дискретных задач показала и теория вычислительной сложности алгоритмов [50, 51, 1, 52]. В соответствии с этой теорией оценка эффективности производится по наихудшему случаю. Так как для большинства дискретных задач оптимизации в наихудшем случае не известен точный алгоритм кроме перебора всех или почти всех вариантов, то полученные оценки являются неудовлетворительными с точки зрения практического использования.

При построении расписания работы детерминированной системы обслуживания актуальна задача понижения размерности, состоящая в преобразовании исходной системы путем группирования требований и назначения укрупненным требованиям всех тех характеристик, которые содержала исходная система обслуживания. Такое преобразование системы называется *агрегирование*, чему в настоящей главе уделено наибольшее внимание.

Понижение размерности системы приводит также к уменьшению накладных расходов на организацию обслуживания, которые, например, согласно некоторым исследованиям [53] в вычислительных системах реального времени достигают 70% (в отдельных случаях и выше) времени работы центрального процессора.

Кроме методов агрегирования, в последнее время все большую популярность стали получать методы параллельного составления расписаний, когда одни параллельные вычислительные системы используются для составления расписания заданий для других параллельных

систем. На данный момент существует достаточно большое количество различных архитектур параллельных вычислительных систем, и каждая из них требует специально адаптированных алгоритмов для эффективного использования. В данной главе разработан алгоритм для одной из параллельных архитектур – параллельный вычислительный кластер.

При исследовании задачи мы полагаем, что на процесс составления расписания влияют ограничивающие факторы возможностей вычислительной системы, на которой осуществляется упорядочение. Основными ограничивающими факторами являются время выполнения процесса составления расписания и объем используемой вычислительной памяти. Будем называть задачами большой размерности те, решение которых нарушает хотя бы одно из ограничений используемой вычислительной системы.

В настоящей главе решаются следующие задачи:

- построение математических моделей многопроцессорных систем без дополнительных ограничений по ресурсам, выполнение заданий в которых осуществляется без прерываний;

- разработка и анализ точных и эвристических алгоритмов решения задачи поиска оптимального расписания в этих системах;

- разработка подхода к решению задачи составления расписания путем агрегирования заданий системы и решения полученной таким образом задачи меньшей размерности;

- разработка параллельных алгоритмов решения задачи составления расписания большой размерности с высоким коэффициентом эффективности и линейной функцией изоэффективности;

- получение аналитических ограничений на точность расписания, получаемого эвристическими алгоритмами, и на время выполнения алгоритмов, разработанных в данной главе;

- подтверждение полученных результатов экспериментальными данными и создание банка вычислительных результатов, которые могут быть в дальнейшем использованы другими исследователями для сравнения практической эффективности новых алгоритмов.

В разд. 2.1 дается формальная постановка задачи. Формализуются термины, которые используются далее в работе. В разд. 2.2 приво-

дятся исследования рассматриваемого и смежных классов задач, проведенные другими учеными. Проводится анализ описанных в литературе алгоритмов, которые могут использоваться для рассматриваемого класса задач, приводятся их сильные и слабые стороны. Выделен ряд наиболее перспективных алгоритмов и методик, которые были использованы для сравнения с разработанными в данной главе алгоритмами. Разд. 2.3 посвящен алгоритмам, разработанным в данной главе: их формальному описанию, определению алгоритмической сложности. В разд. 2.4 описан метода агрегирования для задач построения расписаний. В разд. 2.5 для разработанных приближенных алгоритмов определяется их гарантированная точность. В разд. 2.6 исследуется параллельный процесс построения расписания. В разд. 2.7 собраны и структурированы все полученные вычислительные результаты, приводится описание системы, которая использовалась для проведения экспериментов. Доказательство приведенных утверждений содержится в [42, 104, 105, 108, 109].

2.1. Основные понятия и формальная постановка задачи

2.1.1. Модель

Модель процесса упорядочения, в терминах которой формулируются все последующие задачи, представляется в виде совокупности моделей, описывающих ресурсы и систему заданий. В рассматриваемой модели ресурсы состоят из набора процессоров $P=\{P_1,...,P_m\}$. В зависимости от особенностей задачи они являются либо идентичными, либо одинаковыми только по функциональным возможностям, но разными по быстродействию, либо разными как по возможностям, так и по быстродействию. Общая система заданий для заданного набора ресурсов может быть определена как система $S = \{T, \|\tau_{ij}\|\}$ следующим образом: $T = \{T_1, \ldots, T_n\}$ есть набор работ, подлежащих выполнению; $\|\tau_{ij}\|$ представляет собой целочисленную матрицу размера $n \times m$, элемент которой $\tau_{ij} > 0$ есть время выполнения работы T_i $(1 \le i \le n)$ на процессоре P_j $(1 \le j \le m)$. Будем полагать, что $\tau_{ij} = \infty$, если работа T_i не может быть выполнена на процессоре P_j, и что для каждо-

го i существует, по крайней мере, одно j, для которого $\tau_{ij} < \infty$. В случае, когда все процессоры идентичны, τ_i обозначает время выполнения задания T_i на любом процессоре. Будем рассматривать один из основных показателей эффективности, а именно *длину расписания*, или *максимальное время завершения* : $B = \max\limits_{1 \le j \le m}\{f^j(S)\}$, где $f^j(S)$ – сумма длительностей работ назначенных на j-й процессор.

Основная проблема, таким образом, заключается в нахождении эффективных алгоритмов, позволяющих находить среди всех расписаний такие, для которых эта величина достигает минимума.

2.1.2. Оценка погрешности и времени составления расписания

Для задач, оптимальное решение для которых не известно, использовалась аналитическая нижняя оценка длительности расписания:

$$\underline{B} = \max\left\{\max_i\left(\min_j \tau_{ij}\right);\ \frac{\sum\limits_i \min\limits_j \tau_{ij}}{m}\right\}$$

2.1.3. Задачи большой размерности

В [54] вводится следующее понятие задач большой размерности, которое применяется и в этой работе. Пусть S – множество параметров конкретной задачи (число работ, процессоров, длительности работ). Обозначим через $T(S)$ время решения этой задачи на конкретной вычислительной системе, $V(S)$ – память, необходимая для решения, T_0 и V_0 – соответственно время и память, выделенные для решения задачи. Будем говорить, что рассматриваемая задача является *задачей большой размерности*, если не выполнено хотя бы одно из неравенств.

$$T(S) \le T_0, \qquad V(S) \le V_0.$$

В работе [55] также проводится анализ необходимых вычислительных ресурсов (время, память) при решении задач большой размерности на последовательных вычислительных машинах. В работах [56 - 58] проведено большое исследование, посвященное выявлению задач большой размерности и их параметризации.

2.2. Обзор существующих точных и приближенных алгоритмов

Случайный и исчерпывающий поиск. В книгах [59 - 61] демонстрируется применение достаточно эффективных алгоритмов исчерпывающего поиска к задаче составления расписаний, однако эти алгоритмы перестают быть применимыми при увеличении размерности задачи.

Известны следующие алгоритмы направленного случайного поиска без самообучения [62]: алгоритм с парной пробой, алгоритм с возвратом при неудачном шаге, алгоритм с пересчетом при неудачном шаге, алгоритм с линейной экстраполяцией, алгоритм наилучшей пробы, алгоритм статистического градиента. Обзор некоторых известных методов поиска, а также результаты некоторых алгоритмов приведены в работах [63 - 65].

Математическое программирование. Математическое программирование – это семейство методов оптимизации функций с независимыми ограничениями. Однако этот метод применим только для задач малой размерности. Примеры таких подходов, существующих для задач составления расписаний, можно найти в [66] (линейное целочисленное программирование) и [67].

Динамическое программирование. Основная идея этого метода заключается в замене одновременного выбора большого количества параметров системы поочередным их выбором. Многомерная задача оптимизации сводится к многошаговой задаче меньшей размерности. В основу метода положены принципы оптимальности и инвариантного погружения [68]. Для этого метода также показана его неэффективность для случая задач большой размерности [69].

Метод ветвей и границ. Впервые метод ветвей и границ был предложен Ленд и Дойг [70] в 1960 г. для решения общей задачи целочисленного линейного программирования. Интерес к этому методу и фактически его “второе рождение” связано с работой Литтла, Мурти, Суини и Кэрел [71], посвященной задаче коммивояжера [72].

Быстрые эвристические алгоритмы. В основу быстрых эвристических алгоритмов легли наборы практических методов – *правил*, применяя которые на каждом этапе работы алгоритма можно полу-

чить расписание. Большое исследование, посвященное различным быстрым эвристикам и их применению для широкого круга задач дискретной оптимизации, проведено в [2]. В книге [73] классифицируются более 100 различных правил распределения работ и проведена попытка проанализировать основную идею, лежащую в основе различных правил. Наиболее известными таких правил являются: *первой назначается работа с наименьшей длительностью*, *первой назначается работа с наибольшей длительностью*.

Существуют и более сложные современные эвристические алгоритмы, такие как, например, *муравьиные алгоритмы* [74].

Метод имитации отжига [75, 76]. В основу метода имитации отжига (МИО) лег физический процесс. Отжиг – это термический процесс получения низкоэнергетических уровней твердого тела в тепловой ванне. Процесс состоит из двух этапов: сначала проводится увеличение температуры в ванне до момента плавления твердого тела, затем медленное понижение температуры до состояния, когда частицы тела самоупорядочиваются на основном энергетическом уровне. Фактически МИО – это метод локального поиска, при котором выбирается направление дальнейшей работы. Вместо постоянного поиска наилучшего направления для улучшения решения, МИО изначально выбирает случайное (или полуслучайное) направление, но со временем приходит к наилучшему. Таким образом, процесс выбора наилучшего направления контролируется неким параметром, по аналогии с физическим процессом, называемым *температурой*.

Поиск с запретами (Табу-поиск). Описание процедуры табу-поиска дано в [77] следующим образом: "Метод табу-поиска подразумевает под собой не обращение к сверхчеловеческим силам или моральным предпосылкам, а, наоборот, направлен на выработку и применение ограничений на процесс поиска решений по отношению к областям, попадание в которые привело бы к трудностям. Эти ограничения принимают разные формы. Это может быть исключение определенных альтернатив поиска, классифицированных как "запрещенные", или выработка модифицированных оценок решения и вероятностей выбора". Примеры применения данного метода к задаче составления расписаний можно найти в [78].

Нейронные сети. Этот подход выглядит весьма многообещающим, однако на данный момент фактических результатов в данной области получено немного. Общий подход заключается в использовании нейронных сетей с обратной связью для комбинаторных оптимизационных задач. Так, например, в [79] обсуждается способ применения нейронных сетей для решения задачи обхода доски конем. Некоторые сети были успешно применены для решения задачи коммивояжера. Есть примеры применения нейронных сетей для частных случаев задачи составления расписаний [80, 81], однако количество ограничений при этом возрастает экспоненциально с увеличением размера задачи.

Генетические и эволюционные алгоритмы. Большая работа по исследованию генетических алгоритмов (ГА) и эволюционных алгоритмов (ЭА) проведена в [34] и [35]. Общая схема этих алгоритмов может быть представлена следующим образом [82 - 84]. Основные операции алгоритма: селекция, скрещивание и мутация выполняются над элементами популяции. Результатом их выполнения является очередная популяция. Данный процесс продолжается итерационно до тех пор, пока не будет достигнут критерий останова.

В [85] рассматривается пример использования ГА для проектирования контроллеров физических роботов. Возможность обучения нейронных сетей с помощью ГА также широко обсуждается в [86]. Также встречается большое число примеров использования ГА для решения проблемы составления расписаний [87 - 89]. В [90] рассматривается применение ГА для решения задачи составления расписания экзаменов в университетах.

2.3. Алгоритмы составления расписания без прерываний

В работе будут использоваться следующие сокращения: алгоритм «Процессор с ранним окончанием первым» – ПРОП, вероятностный алгоритм – ВА, псевдополиномиальный алгоритм с поиском в глубину – ПАПГ, псевдополиномиальный алгоритм с поиском в ширину – ПАПШ, алгоритм «Самая длинная работа первой» – СДРП, алгоритм «Самая короткая работа первой» – СКРП, комбинированный агреги-

рующий алгоритм – КАА, переборный агрегирующий алгоритм – ПАА, многоуровневый агрегирующий алгоритм – МАА, комбинированный псевдополиномиальный алгоритм – КПА.

2.3.1. Алгоритм «Процессор с ранним окончанием первым»

Процедура работы этого алгоритма состоит в следующем. На k-м шаге распределяемая работа (ее индекс k) назначается на тот процессор, суммарное время выполнения работ на котором с учетом данной работы минимальное. Иными словами, минимизируется по j выражение $(B_{k-1}^{j} + \tau_{kj})$, где B_{k-1}^{j} – суммарное время выполнения работ, назначенных на j-й процессор на первых k-1 шагах. Алгоритмическая сложность рассматриваемой эвристики составляет $O(nm)$. Это следует из того, что на каждом из n шагов (по числу работ) необходимо произвести m сложений (для определения времен окончания) и m-1 сравнений (для определения минимума этих времен).

2.3.2. Вероятностный алгоритм

В [100] предложен метод решения задачи упаковки, состоящий в нахождении приближенного решения задачи целочисленного программирования путем решения релаксационной задачи линейного программирования, а затем округления полученного решения. Для некоторых задач упаковки с помощью вероятностного метода доказано существование целочисленного решения, близкого к оптимуму релаксационной задачи, и предложен детерминированный алгоритм построения аппроксимированного решения. Автор назвал разработанный метод методом аппроксимации решеткой.

Математическая основа предложенного алгоритма заключается в следующем. Пусть дана матрица C, элементы которой $c_{ij} \in [0,1]$, и вектор $p = (p_1, p_2, \ldots, p_m)$, $p_j \in \Re$, i=1,…,n; j=1, …, m. Требуется найти такой целочисленный вектор $q = (q_1, q_2, \ldots, q_m)$ (узел решетки), который «хорошо» аппроксимирует p в том смысле, что по всем координатам величина $\left| C \bullet (p - q) \right|$ мала. Таким образом, задача сводится к вычислению вектора q при ограничениях сверху на величину

$\Delta_i = \left| \sum_{j=1}^{m} c_{ij}(p_j - q_j) \right|$. Без ограничения общности можно полагать, что $p_j \in [0,1]$ при всех j. Рассмотрим подробнее работу вероятностного алгоритма. Без ограничения общности будем полагать, что в выражении $\Delta_i = \left| \sum_{j=1}^{m} c_{ij}(p_j - q_j) \right|$ значения p_j для всех j лежат на отрезке [0, 1]. Действительно, при произвольных значениях p_j мы можем вычесть из него целую часть, и работать только с остатком (дробной частью). Тогда будем рассматривать q_j как округленные p_j. Для этого обратимся к методу, называемому *методом случайного округления*. Будем полагать q_j равным единице с вероятностью p_j независимо от значений других компонент вектора q. В [100] доказывается, что существует такой вектор q, что $\Delta_i \le s_i D(s_i,\ 1/2n)$, где $s_i = \sum_{j=1}^{m} c_{ij} p_j$, а функция D удовлетворяет условиям: $D(m,\ x) \le (e-1)\left[\frac{\ln 1/x}{m}\right]^{1/2}$ при $m > \ln 1/x$; и $D(m,\ x) \le \frac{e \ln 1/x}{m \ln[(e \ln 1/x)/m]}$ при $m \le \ln 1/x$. Там же приводится алгоритм (*названный методом условных вероятностей*) построения такого вектора.

Рассмотрим подробнее работу предложенного метода для рассматриваемой задачи. Запишем задачу в следующем виде.

$$\sum_{j=1}^{m} x_{ji} = 1,\ i = 1,\ \ldots, n, \tag{1}$$

$$\sum_{i=1}^{n} x_{ji} \tau_{ij} \le B,\ j = 1, \ldots, m, \tag{2}$$

$$x_{ji} \in \{0,1\}, \tag{3}$$

$$Z = B \to min. \tag{4}$$

Поясним смысл такой интерпретации. Величина $x_{ji} \in \{0,1\}$ есть показатель того, будет ли выполнена работа T_i на процессоре P_j. Так, если $x_{ji} = 1$, то в соответствующем расписании работа T_i будет выполнена на процессоре P_j, если $x_{ji} = 0$, то работа T_i не будет выполняться на процессоре P_j. При этом условие (1) означает, что каждая работа должна быть выполнена, а наложение условия целочисленности на x_{ji}

означает, что работа будет выполнена только на одном процессоре (расписание без прерываний). Условие (2) – это ограничение на длину расписания, а условие (4) означает, что задача состоит в минимизации длины расписания.

Отметим, что вместо вектора p в данной формулировке участвует матрица с элементами x_{ji}, что, однако, не влияет на общий смысл работы алгоритма. Сначала найдем решение релаксационной задачи линейного программирования, в которой $x_{ji} \in [0, 1]$, например, с помощью симплекс-метода. Пусть получена матрица с элементами x_{ji}^*. Затем найдем такую целочисленную матрицу $\bar{x}$, аппроксимирующую x_{ji}^*, что $P(\bar{x}_{ji} = 1) = x_{ji}^*$ и выполняется условие (1). Это можно сделать следующим образом. Будем заполнять элементы матрицы $\|\bar{x}_{ji}\|$ по строкам: для каждого j будем брать последовательно элементы $\bar{x}_{ji}$ с i от 1 до n, полагая $\bar{x}_{ji}$ равным 1 с вероятностью x_{ji}^*. Если некоторый элемент $\bar{x}_{ji}$, полученный таким образом, окажется равным 1, то остальные элементы строки (элементы с тем же значением j) полагаем равными 0.

Определим алгоритмическую сложность этого алгоритма. Алгоритмическая сложность решения релаксационной задачи в такой постановке есть $O(nm)$. В процессе определения процессора, на который будет назначена работа T_i, проводится m независимых экспериментов со случайным исходом: $\bar{x}_{ji}$ присваивается значение 1 с вероятностью x_{ji}^*. Вероятность того, что в результате серии из m экспериментов ни один из $\bar{x}_{ji}$ не будет равен 1, равна

$$\bar{P}_1(1) = \prod_j \left(1 - x_{ji}^*\right) \le \prod_j \left(1 - \frac{1}{m}\right) = \left(1 - \frac{1}{m}\right)^m.$$

Вероятность того, что за k проходов ни одна работа не будет назначена, равна

$$\bar{P}_1(k) = \left(\prod_j \left(1 - x_{ji}^*\right)\right)^k \le \left(1 - \frac{1}{m}\right)^{mk}.$$

Тогда вероятность того, что за некоторое количество проходов k все n работ будут распределены (для каждого i будет такое j, что $\bar{x}_{ji} = 1$), равна

$$P_n(k) = \left(1 - \left(\prod_j \left(1 - x_{ji}^*\right)\right)^k\right)^n \geq \left(1 - \left(1 - \frac{1}{m}\right)^{mk}\right)^n.$$

На рис. 1 приведен фазовый потрет функции вероятности $P_n(k)$. Как видно из графиков при значениях k около 10 вероятность назначения всех работ достаточно велика, что хорошо коррелирует с результатами вычислительных экспериментов.

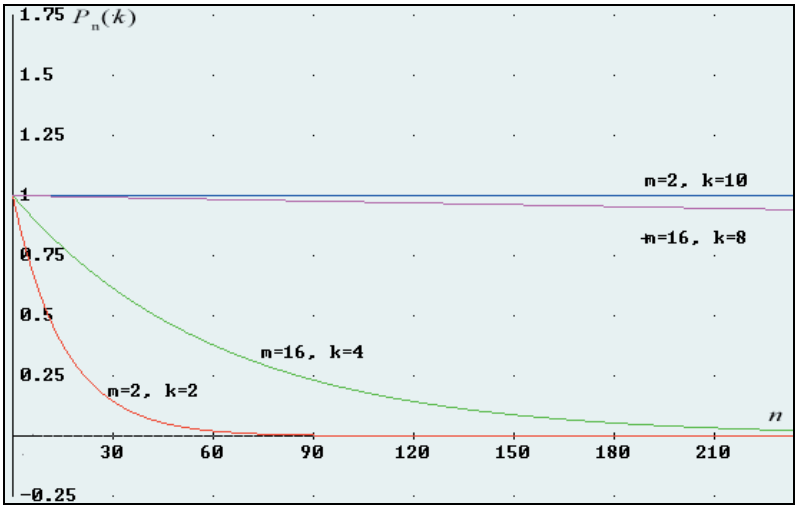


Рис. 1. Фазовый потрет $P_n(k)$

Таким образом, с вероятностью $P_n(k)$ полная алгоритмическая сложность ВА есть $O(nmk)$.

2.3.3. Псевдополиномиальные алгоритмы

2.3.3.1. Различные интерпретации псевдополиномиальных алгоритмов

Дадим следующую интерпретацию рассматриваемой задачи. Необходимо решить оптимизационную задачу вида $\left\| \sum_{i=1}^{n} \vec{p}_i \right\| \to \min$, где

$\vec{p}_i = (0,...,\tau_{ik},...,0)$ – m-мерный вектор. Для произвольного вектора $\vec{a} = (a_1, a_2, ..., a_m)$ $\left\|\vec{a}\right\| = \max_k a_k$ – величина максимальной компоненты. В теории оптимизации задача в подобной формулировке называется «задачей об упаковке». Однако несложно заметить, что «задача об упаковке» имеет однозначное соответствие с рассматриваемой задачей: достаточно провести соответствие вектора $\vec{p}_i = (0,...,\tau_{ik},...,0)$ с работой T_i, где k – номер процессора, на который назначена эта работа. Суть псевдополиномиального алгоритма состоит в последовательном построении и переборе вариантов с дополнительным условием отсева, при котором из рассмотрения выбрасываются варианты, в процессе построения которых сумма векторов превысила по какой-либо из

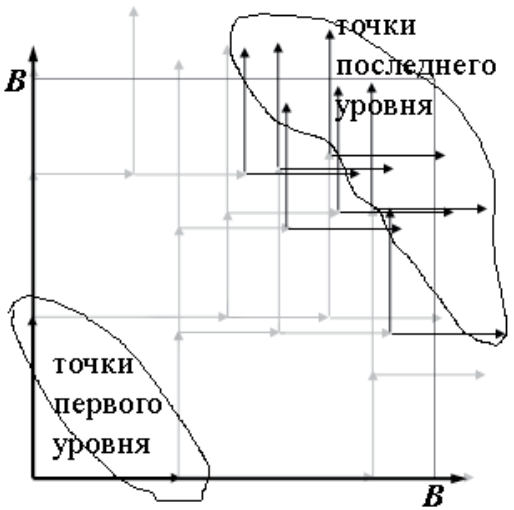


Рис. 2. Векторная интерпретация псевдополиномиального алгоритма

из компонент определенный заранее директивный интервал.

Другая интерпретация алгоритма связана с построением дерева расписаний. Вершины дерева в такой интерпретации соответствуют работам, а ребра дерева – назначению работы на процессор. При этом весу каждого ребра дерева ставится в соответствие длительность выполнения определенной работы на соответствующем процессоре. Так, например, в случае, рассматриваемом на рис. 3, вершина с порядко-

вым номером генерирования 1 соответствует назначению первой работы на первый процессор, а вершины с порядковым номером 2 соответствует расписанию, при котором первая вершина назначена на второй процессор. Веса ребер равны соответственно τ_{11} и τ_{12}. При генерации новой вершины проверяется условие, согласно которому сумма весов «сонаправленных» ребер по полученной ветви дерева не должна превышать директивного интервала. Если это условие нарушается, то соответствующая вершина не создается. На рис. 3 числа в левой части вершины означают порядок, в котором вершины были сгенерированы, в правой – порядок, в котором они использованы для дальнейшего ветвления.

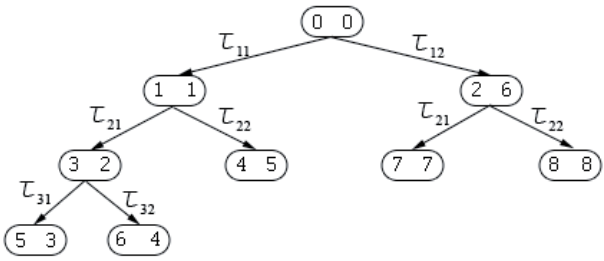


Рис. 3. Интерпретация в виде дерева

2.3.3.2. Псевдополиномиальный алгоритм с поиском в ширину

Работу алгоритма, решающего поставленную задачу, опишем следующим образом. Вычислим величину *B*, которую будем называть директивным сроком. Число *B* можно получить, например, с помощью описанных выше алгоритмов ПРОП или ВА. Затем будем рассматривать *m*-мерный куб со стороной *B* (в векторной интерпретации). Необходимо выбрать вектора $\vec{p}_i$ так, чтобы каждая компонента вектора $\sum_{i=1}^{n} \vec{p}_i$ не превосходила *B*. Под выбором вектора $\vec{p}_i$ будем понимать выбор процессора, на который назначена работа T_i, что однозначно задает вектор. Выбор осуществляем путем построения множества то-

чек m-мерного пространства, для каждой из которых на l-м уровне проверяем, принадлежит ли точка m-мерному кубу со стороной B (рис. 2). Если условие не выполнено, то точка исключается из дальнейшего рассмотрения. Формализуем алгоритм по шагам.

1. Строим первые m точек m-мерной решетки: $(\tau_{11}, 0, \ldots, 0)$, $(0, \tau_{12}, \ldots, 0)$, …, $(0, 0, \ldots, \tau_{1m})$. Полагаем $l = 1$, а все указанные точки активными.

2. Исключаем из списка активных точек те, которые не принадлежат m-мерному кубу со стороной B. Если $l = n$, то переходим к п. 4. В противном случае переходим к п. 3.

3. Полагаем $l = l + 1$. Каждую активную точку $(a_1, a_2, \ldots, a_m)$, полученную на шаге $l - 1$, исключаем из списка активных точек и строим новые активные точки $(a_1+\tau_{l1}, a_2, \ldots, a_m)$, $(a_1, a_2+\tau_{l2}, \ldots, a_m)$, …, $(a_1, a_2, \ldots, a_m+\tau_{lm})$. Переходим к п. 2.

4. Множество решений найдено. Из числа активных точек находим точку, соответствующую оптимальному решению. Завершаем работу алгоритма.

Таким образом, в результате выполнения алгоритма определяются все расписания с длиной, не превосходящей B, и из них выбирается оптимальное. Отметим, что для шага 2 важно, чтобы множество активных точек не было пусто, что верно, т.к. существует хотя бы одно решение, найденное с помощью алгоритма ПРОП.

Из описания алгоритма видно, что на первом шаге выполняется $O(m)$ действий с точками m-мерного пространства (всего $O(m^2)$ операций), затем на этапах 2 и 3 алгоритма для каждой из активных точек m-мерного пространства выполняется по $O(m)$ операций. Поскольку в худшем случае число вершин ветвления совпадает с числом узлов целочисленной решетки в m-мерном кубе со стороной B (т.е. с числом $(B+1)^m$ в силу целочисленности задачи), то вычислительная сложность пунктов 1-3 алгоритма составляет $O(m^2(B+1)^m)$. В пункте 4 необходимо из множества концевых точек допустимых решений найти такую $(a_1, a_2, \ldots, a_m)$, для которой $\max\limits_k a_k$ минимально. Поскольку для каждой оконечной точки необходимо выполнить $(m\text{-}1)$ сравнений, то окончательная вычислительная сложность алгоритма есть $O(m^2B^m)$.

При фиксированном числе процессоров m алгоритм является псевдополиномиальным с алгоритмической сложностью $O(B^m)$.

2.3.3.3. Псевдополиномиальный алгоритм с поиском в глубину

Псевдополиномиальный алгоритм с поиском в глубину отличается от рассмотренного в предыдущем разделе алгоритма следующим. Вместо построения полного дерева решений, укладывающегося в m-мерный куб со стороной B, находится одно решение, удовлетворяющее директивному интервалу. Затем производится уточнение директивного интервала, например путем половинного деления. Начальный интервал возможных значений B: ($\underline{B}$, $\overline{B}$), где

$$\underline{B} = \max\left\{\max_i\left(\min_j \tau_{ij}\right);\ \frac{\sum \min_j \tau_{ij}}{m}\right\},$$ а $\overline{B}$ – значение, полученное каким-либо быстрым алгоритмом (например, ПРОП или ВА). Затем вычисляется величина $B_1=(\overline{B}+\underline{B})/2$, которая используется в качестве следующего значения директивного срока. В случае если решение с таким директивным сроком найдено, то директивный срок для последующей итерации вычисляется по формуле $B_2=(B_1+\underline{B})/2$, в обратном случае по формуле $B_2=(\overline{B}+B_1)/2$, и т.д. Таким образом, этот алгоритм является итерационным. В худшем случае каждая итерация алгоритма может иметь такую же сложность, как и алгоритм с поиском в ширину, однако в среднем каждая итерация этого алгоритма работает существенно быстрее (это подтверждается экспериментальными данными). Представленный алгоритм обладает также тем преимуществом, что с его помощью можно находить как точные решения (для целочисленных задач), так и приближенные решения с заданной точностью. Для этого следует остановить работу алгоритма при разности между верхней и нижней оценками, не превосходящей требуемой точности.

Аналитические результаты по алгоритмической точности алгоритма и ее связи с точностью получаемого решения приведены в разделе 2.5.

2.4. Метод агрегирования

Как уже отмечалось выше, высокая временная сложность переборных алгоритмов не дает возможности использовать их в существующих вычислительных машинах (а особенно в вычислительных системах реального времени) при достаточно большом числе заданий, и даже чрезвычайно высокие темпы развития компьютерных технологий не позволяют надеяться на создание в ближайшем времени соответствующих вычислительных ресурсов. Метод агрегирования позволяет существенно уменьшить перебор.

2.4.1. Формальное описание

Элементарным агрегированием некоторой системы $S = \{T, \|\tau_{ij}\|\}$ ($i=1,\ldots,n$) называется переход к системе $S' = \{T', \|\tau_{kj}'\|\}$ ($k = 1, \ldots, l$) полученной путем объединения заданий T_i, $T_j \in T$ в задание $T_k' \in T'$ с характеристиками, не противоречащими характеристикам объединяемых заданий.

Под непротиворечивостью характеристик систем S и S' будем понимать следующее: $\sum_{i=1}^{n}\tau_{ij} = \sum_{k=1}^{l}\tau'_{kj}$ при при всех j. Переход к некоторой системе $S' = \{T', \|\tau_{kj}'\|\}$ путем последовательности элементарных агрегирований системы $S = \{T, \|\tau_{ij}\|\}$ назовем агрегированием системы S.

Таким образом, модель создания агрегированной задачи можно описать следующим образом.

1. С помощью некоторого алгоритма проводим декомпозицию набора заданий исходной системы T на несколько меньших наборов T^{*i} $(1 \le i \le s)$, $\bigcup T^{*i} = T$. Получим набор систем $S^{*i} = \{T^{*i}, \|\tau_{kj}{}^{*i}\|\}$.
2. Для каждой из систем S^{*i} с помощью точного (или приближенного) алгоритма находится точное (приближенное) решение.
3. На основании полученных в п.2 последовательностей выполнения заданий проводим агрегирование исходной системы S, получаем систему S' такую, что любое из заданий $T_k' \in T'$ пред-

ставляет собой набор последовательно выполняемых работ $T_i \in T$.

4. Для системы S' с помощью точного (или эвристического) алгоритма находим точное (приближенное) решение.
5. Если достигнута требуемая точность (или выполнен другой критерий останова), то переходим к п. 7.
6. Повторяем операции пунктов 1 – 5 (N-1)-раз, где N будем называть глубиной агрегирования. Получим некоторую систему $S'^{(N)}$.
7. Окончательным решением будем считать представление решения задачи для системы $S'^{(N)}$, в котором вместо заданий $T_k'^{(N)} \in T'^{(N)}$ представлены задания из набора исходной системы.

Другими словами этот подход содержит следующие основные элементы: последовательное разбиение задачи на подзадачи упорядочения существенно меньшей размерности, формирование дерева подзадач, решение подзадач, формирование решения задачи из решения подзадач в соответствии с построенным деревом подзадач.

На рис. 4 показан пример построенного дерева подзадач для задачи упорядочения 22 работ по 4 процессорам с глубиной агрегирования N=2 (Рис. 4а) и N=1 (Рис. 4б). Кружочки на рисунке символизируют работы, прерывистые линии – назначение работ на процессоры с помощью некоторого алгоритма (точного или приближенного), сплошные линии – процесс агрегирования, прямоугольники – подзадачи, количество частей в них соответствует количеству процессоров.

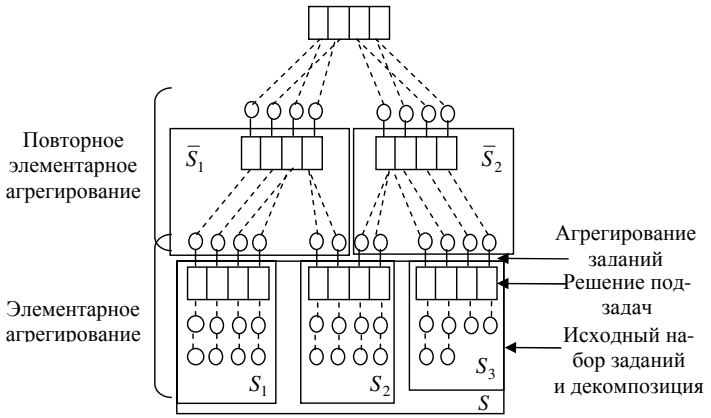


Рис 4а.

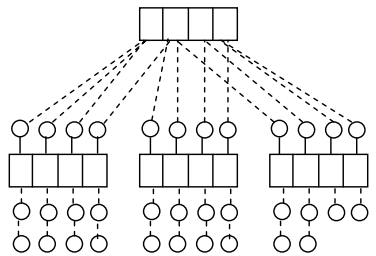


Рис 4б.

2.4.2. Задача составления расписания *n* работ на *m* идентичных процессорах

При применении исследуемой модели к данной задаче использовались следующие методы. Два метода с эвристическими алгоритмами составления расписания: СДРП и СКРП, и три метода с применением агрегирования, характеризовать которые будем по глубине агрегирования и применяемым алгоритмам упорядочения. Метод с переборным алгоритмом – ПАА (глубина агрегирования N=1, алгоритмы в подзадачах и в агрегированной задаче – точные), с комбинированным алгоритмом – КАА (N=1, алгоритм в подзадачах точный, в агрегированной задаче приближенный), и многоуровневый алгоритм – МАА (N не фиксировано, алгоритмы точные).

Будем использовать методику декомпозиции, разработанную в [101], и адаптируем ее для рассматриваемой задачи. Пусть $N = \{1, 2, \ldots, n\}$ – конечное множество работ. Каждой паре $i \in N$ и $j \in N$ $(i \neq j)$ поставлено в соответствие число $c_{ij} \geq 0$, называемое разностью. Величина $c_{ij} = |\tau_i - \tau_j|$. Числа c_{ij} могут быть элементами матрицы расстояний $C = (c_{ij})_{n \times n}$. В этом случае задача поставлена в матричной форме.

Рассмотрим задачу о разбиении множества N на непересекающиеся подмножества $S_1, S_2, \ldots, S_k$, $k \geq 2$:

$$\bigcup_{i=1}^{k} S_i = N;$$

$$S_i \bigcap S_j = \varnothing, i \neq j, i, j = 1, 2, \ldots, k;$$

$$0 < n_{\min} \leq |S_i| \leq n_{\max}, i = 1, 2, \ldots, k.$$

Число подмножеств k определяется так, чтобы выполнялось условие:

$$k \times n_{\min} \leq n \leq k \times n_{\max}.$$

Каждому разбиению $S=(S_1,S_2,\ldots,S_k)$, удовлетворяющему указанным условиям, поставим в соответствие значение функции $\varphi(S) = \varphi(S_1, S_2, \ldots, S_k)$, характеризующей качество разбиения. Необходимо найти разбиение $S^0 = (S^0_1, S^0_2, \ldots, S^0_k)$, такое что

$$\varphi(S^0) = \min_S \varphi(s).$$

Рассмотрим способы вычисления значений функции $\varphi(s)$. Предположим, что известны функции $g_p(S_i)$, характеризующие различие работ, принадлежащих подмножеству S_i. В качестве функций $\varphi_p(S)$ при фиксированном значении p рассмотрим одну из двух функций:

$$\varphi_p(S_1, S_2, \ldots, S_k) = \sum_{i=1}^{k} g_p(S_i),$$

$$\varphi_p(S_1, S_2, \ldots, S_k) = \max_{1 \leq i \leq k} g_p(S_i),$$

где способы вычисления значений $g_p(S_i)$ следующие:

а) для каждого $j \in S_i$ находим $r(j) \neq j$, такое, что

$$c_{j,r(j)} = \min_{r \in S_i} c_{jr}, \; g_1(S_i) = \sum_{j \in S_i} c_{j,r(j)};$$

б) $g_2(S_i) = \max c_{ij}$;

в) $g_3(S_i) = \max c_{tj} - \min c_{tj}$;

г) $g_4(S_i) = \dfrac{\max c_{tj}}{\min c_{tj}}, \min c_{tj} > 0; j \in S_i, t \in S_i, j \neq t.$

В данной работе применялся алгоритм, удовлетворяющий следующим условиям:

1. Времена решения каждой из подзадач должны быть примерно равны, т.о. количество работ в каждой из подзадач должно быть примерно одинаковым.

2. Длительности работ в одной подзадаче должны быть близки:

   $\delta = \sum_{i=1}^{k} \max_{T_l, T_p \in S^i} | \tau_l - \tau_p |$ мало.

3. Время проведения декомпозиции достаточно мало.

Работы сортировались по длительности. Полученный таким образом набор разбивался на равные по количеству работ поднаборы (по числу подзадач, которое являлось параметром декомпозиции). В случае, если количество работ в задаче не было кратно числу подзадач, работы «на стыке» добавлялись в поднабор в соответствии с минимизацией δ.

2.4.3. Вспомогательные алгоритмы

Алгоритм «Самая длинная работа первой». Выбирается задача с наибольшей длительностью и назначается на процессор, сумма времен выполнения задач на котором наименьшая; если таких процессоров несколько, выбирается любой из них; затем эта задача исключается из списка рассматриваемых. После этого из списка оставшихся задач снова выбирается задача с наибольшей длиной и назначается на наименее загруженный процессор; операция повторяется до тех пор, пока список активных задач не окажется пустым.

Этот метод является одним из наиболее быстрых эвристических методов решения задачи упорядочения и имеет алгоритмическую сложность $O(n \log n)$. Покажем это. Первым шагом алгоритма отсортируем набор работ по убыванию. Время работы алгоритма сортировки – $O(n \log n)$. Затем произведем «связывание» процессоров в одно-

направленный список так, что наименее нагруженный процессор находится на вершине списка. Время работы алгоритма «связывания» – $O(m)$. Работа алгоритма заключается в следующем: из набора отсортированных работ берется верхняя (самая длинная) и помещается на процессор, находящийся на вершине списка процессоров (наименее загруженный), затем этот процессор меняет свое положение в списке в зависимости от текущей загруженности. Его местоположение может быть определено, например, методом половинного деления. Таким образом, общее время работы алгоритма составит $O(n \log n) + O(m) + O(n \log m)$. Поскольку можно считать, что $m < n$, то алгоритмическая сложность составляет $O(n \log n)$.

Алгоритм «Самая короткая работа первой». Выбирается задача с наименьшей длительностью и назначается на процессор, сумма времен выполнения задач на котором наименьшая; если таких процессоров несколько, выбирается любой из них; затем эта задача исключается из списка рассматриваемых. После этого из списка оставшихся задач снова выбирается задача с наименьшей длиной и назначается на наименее загруженный процессор; операция повторяется до тех пор, пока список активных задач не окажется пустым. Из описания видно, что алгоритм СКРП аналогичен СДРП и отличается лишь направлением сортировки множества работ. Однако, как показывают проведенные эксперименты, результаты работы этих алгоритмов существенно различаются. Алгоритмическая сложность данного алгоритма равна $O(n \log n)$.

2.4.4. Агрегирующие алгоритмы

Переборный Агрегирующий Алгоритм. Алгоритм предполагает упорядочение работ в подзадачах и агрегированной задаче путем перебора. Перебор может быть полным, либо осуществляться с помощью метода ветвей и границ. Этот метод имеет одно существенное ограничение. Пусть задачу упорядочения n работ по m процессорам мы хотим разбить на s подзадач. Для того чтобы получить подзадачи заметно меньшей размерности, необходимо брать s большим. Однако при решении агрегированной задачи нам будет необходимо решить

задачу упорядочения ms работ по m процессорам, вычислительная сложность которой может оказаться слишком большой за счет величины s.

Комбинированный Агрегирующий Алгоритм. Процедура применения этого алгоритма следующая. Каждая из подзадач меньшей размерности упорядочивается путем перебора. Решение же агрегированной задачи строится с помощью эвристического алгоритма. Таким образом, решается проблема, возникающая при применении переборного агрегирующего алгоритма.

Многоуровневый Агрегирующий Алгоритм. Проводится декомпозиция задачи упорядочения n работ по m процессорам на $s^{(1)}$ подзадач. После решения подзадач (переборным алгоритмом) и агрегирования получим задачу упорядочения $ms^{(1)}$ работ по m процессорам. Полученная задача снова декомпозируется на $s^{(2)}$ подзадачи, которые решаются аналогично. Когда, после некоторого количества повторений, мы получим задачу упорядочения $ms^{(N\ -\ 1)}$ работ по m процессорам, время решения которой приемлемо, назначение работ проводится в последний раз. Таким образом, в этом методе число управляющих переменных составляет N. Возможность их варьирования позволяет уменьшить погрешность метода, но увеличивает его алгоритмическую сложность. В работе приведены результаты экспериментов, в которых $s^{(0)}=s$, $s^{(i+1)}=[s^{(i)}/2]$.

2.4.5. Анализ возможности совместного использования методики агрегирования с другими алгоритмами

Быстрые эвристические алгоритмы. В работе были рассмотрены следующие быстрые эвристические алгоритмы – СДРП, СКРП и ПРОП. Полученные в работе результаты однозначно показывают неэффективность алгоритма СКРП по сравнению с СДРП для рассматриваемой задачи. Поэтому для задач подобного рода имеет смысл ограничиться лишь использованием алгоритма СДРП.

Алгоритмы СДРП и ПРОП могут совместно использоваться с методом агрегирования для получения первоначальной верхней оценки расписания и задания разбиения множества работ на подмножества, к

которым затем применяется агрегирование (алгоритм СДРП может быть применен только в случае идентичных процессоров). Такое совместное использование было исследовано в работе в качестве алгоритма КПА. Для описанного совместного использования можно считать алгоритм ПРОП более эффективным, т.к. область его применения шире, чем СДРП, причем для случая идентичных процессоров задача может быть адаптирована для получения результатов не хуже, чем при использовании алгоритма СДРП (если набор работ будет предварительно отсортирован по невозрастанию длительностей, то алгоритм ПРОП будет совпадать с СДРП). Как показано выше, время работы алгоритма ПРОП есть $O(nm)$. Таким образом, получение верхней оценки происходит очень быстро, и время работы этого алгоритма полностью «поглощается» другими, более трудоемкими этапами составления расписания. Алгоритм СДРП также может быть использован для «сборки» общего расписания из расписаний агрегированных подзадач.

2.5. Алгоритмы с гарантированной точностью

2.5.1. Псевдополиномиальный алгоритм с поиском в глубину

Лемма 1. С помощью ПАПГ можно найти решение с погрешностью $\varepsilon = \frac{B - B^*}{B^*}$ за время $O\left(\left(\log\left(\frac{\overline{B} - \underline{B}}{\varepsilon \underline{B}}\right) + 1\right) m^2 \left(\overline{B} - \varepsilon \underline{B}\right)^m\right)$, если $\varepsilon \underline{B} > 1$ и за время $O\left(\log\left(\overline{B} - \underline{B}\right) m^2 \overline{B}^m\right)$, если $\varepsilon \underline{B} \le 1$, где B – длина расписания, найденного ПАПГ, B^* – длина оптимального расписания, $\overline{B}$ и $\underline{B}$ – первоначальные верхняя и нижняя оценки.

Лемма 2. За время $O\left(\log\log\left(\frac{\overline{B}}{\underline{B}}\right) m^2 \overline{B}^m\right)$ с помощью ПАПГ можно найти расписание длиной B, такое, что $\frac{1}{2} B \le B^* \le B$, где B^* – длина оптимального расписания.

2.5.2. Вероятностный алгоритм

Лемма 3. С вероятностью $P_n(k) \ge \left(1-\left(1-\frac{1}{m}\right)^{mk}\right)^n$ за время $O(nmk)$ с помощью вероятностного алгоритма можно с вероятностью, большей $1-\frac{1}{n}$, найти решение с погрешностью $\varepsilon = \frac{B-B^*}{B^*} \le D\left(B^*, \frac{1}{n}\right)$, где

$$D\left(B^*, \frac{1}{n}\right) \le (e-1)\sqrt{\frac{\ln n}{B^*}} \text{ при } B^* > \ln n,$$

$$D\left(B^*, \frac{1}{n}\right) \le \frac{e \ln n}{B^* \ln\left(\frac{e \ln n}{B^*}\right)} \text{ при } B^* \le \ln n.$$

2.5.3. Алгоритм «Процессор с ранним окончанием первым»

Лемма 4. Погрешность ε расписания, получаемого алгоритмом ПРОП, составляет $\varepsilon = \frac{B-\underline{B}}{\underline{B}} \le m-1$, где m – число процессоров, B – длина расписания, полученного алгоритмом ПРОП, $\underline{B} = \max\left\{\max_i\left(\min_j \tau_{ij}\right); \frac{\sum_i \min_j \tau_{ij}}{m}\right\}$ – нижняя оценка длины оптимального расписания.

Лемма 5. При решении задачи составления расписания на идентичных процессорах, погрешность расписания ε, получаемого алгоритмом ПРОП, составляет $\varepsilon = \frac{B-B^*}{B^*} \le 1$, где B – длина расписания, полученного алгоритмом ПРОП, B^*– длина оптимального расписания.

2.5.4. Алгоритм «Самая длинная работа первой»

Лемма 6. При решении задачи составления расписания работ на идентичных процессорах с использованием алгоритма СДРП погреш-

ность ε получаемого расписания составляет $\varepsilon = \frac{B - B^*}{B^*} \le 1$, где B – длина полученного расписания, B^* – длина оптимального расписания.

2.6. Параллельное выполнение вычислений

2.6.1. Комбинированный псевдополиномиальный алгоритм

Для решения задачи составления расписаний большой размерности предлагается следующий алгоритм. С помощью эвристического алгоритма строится расписание, длина которого $\overline{B}_0$ берется за оценку сверху длины итогового расписания. На следующем шаге проводится декомпозиция задачи на подзадачи в соответствии с полученным расписанием. Для этого проводится разбиение процессоров на группы с некоторым задаваемым параметром M (M равно количеству групп, на которое разбивается множество процессоров) таким образом, что в первую группу попадает $[m/2M]$ наиболее загруженных процессоров (суммарное время выполнения работ, на которых наибольшее), и столько же наименее загруженных. Аналогично строятся остальные группы из оставшихся процессоров. Затем производится декомпозиция множества работ. Оно также разбивается на M групп, и в каждую группу попадают те работы, которые были назначены эвристическим алгоритмом на процессоры из соответствующей группы. Каждая из полученных таким образом подзадач решается с помощью описанного выше псевдополиномиального алгоритма.

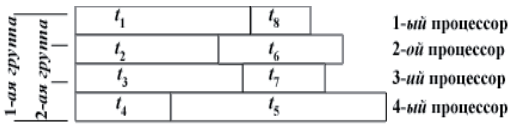


Рис. 5

На рис. 5 схематически изображено распределение 8 работ по 4 процессорам, полученное с помощью алгоритма ПРОП. Если рассмотреть случай декомпозиции исходной задачи на две подзадачи, то первой подзадачей будет «распределить работы t_1, t_8, t_4, t_5 по процессорам 1 и 4», а второй подзадачей – «распределить работы t_2, t_6, t_3, t_7

по процессорам 2 и 3». Каждая из полученных подзадач будет затем решена с помощью псевдополиномиального алгоритма.

Если в исходной задаче требуется получить расписание с заданной точностью, то на первом шаге для построения первоначального решения вместо эвристического алгоритма можно использовать псевдополиномиальный алгоритм с требуемой точностью.

2.6.2. Параллельное построение дерева решения

Для определения эффективности алгоритмов, основанных на параллельном построении дерева решений, рассматривают (например, в исследованиях, проведенных в [91, 96]) следующие величины.

1. Размерность задачи W (для рассматриваемого класса задач построения дерева решений это количество вершин в дереве).

2. Коэффициент ветвления b (среднее число потомков у вершин дерева решений).

3. Размер параллельной вычислительной системы q (число процессоров, используемых для проведения параллельной работы алгоритма).

4. Времена параллельной и последовательной работы алгоритма: T_q (время выполнения алгоритма на q процессорах) и T_1 (время последовательного выполнения алгоритма). Будем предполагать, что T_1 пропорционально W.

5. Время проведения вычислений $T_{\text{выч}}$ (сумма времен, затраченных всеми процессорами для проведения вычислений). Будем предполагать, что количество вычислений при параллельном и последовательном выполнении в среднем одинаково. Поэтому значения величины $T_{\text{выч}}$ для q процессоров и для одного процессора совпадают и равны T_1.

6. Время взаимодействия $T_{\text{вз}}$ (сумма времен, затраченных всеми процессорами на взаимодействие с другими процессорами с целью получения работы (сюда также включается простой в случае, если работы больше нет). Поскольку каждый процессор в любой момент времени либо проводит вычисление, либо ожидает новую работу, то $T_{\text{вз}} + T_{\text{выч}} = q \times T_q$.

7. Прирост в скорости $S = \frac{T_1}{T_q}$ (отношение, показывающее уменьшение времени работы алгоритма при использовании параллельной вычислительной системы).

8. Эффективность E (прирост в скорости, отнесенный к числу параллельных процессоров; демонстрирует эффективность использования параллельной вычислительной системы):

$$E = \frac{S}{q} = \frac{T_1}{T_q \times q} = \frac{T_{\text{выч}}}{T_{\text{выч}} + T_{\text{вз}}} = \frac{1}{1 + \frac{T_{\text{вз}}}{T_{\text{выч}}}}.$$

9. Единица вычислительного времени $U_{\text{выч}}$ (среднее время, затрачиваемое на ветвление одной вершины дерева решений).

10. Единица времени взаимодействия $U_{\text{вз}}$ (среднее время взаимодействия между процессорами вычислительной системы для получения новой работы).

Эффективность и прирост по скорости, достигаемые параллельными вычислительными алгоритмами, определяются архитектурой параллельной вычислительной системы, алгоритмом распределения работ, количеством процессоров и размерностью системы. При заданной размерности системы W увеличение числа процессоров q приводит к уменьшению эффективности в силу того, что $T_{\text{вз}}$ увеличивается, а $T_{\text{выч}}$ остается прежней. При заданном q увеличение W улучшает эффективность (в качестве примера см. кривую прироста скорости для архитектуры Intel Hypercube в [93]). Таким образом, при увеличении q существует способ сохранить значение эффективности фиксированным (т.е. поддерживать линейную зависимость прироста скорости) путем увеличения W. Скорость изменения W по отношению к q зависит от архитектуры и алгоритма распределения работ.

Во многих параллельных алгоритмах (таких, как параллельный алгоритм для задачи о ранце с одним ограничением [91], алгоритмы поиска кратчайшего пути и быстрой сортировки [102]) существует возможность поддерживать линейный прирост скорости при произвольном числе процессоров путем увеличения размерности задачи. Скорость увеличения W по отношению к q (для сохранения фиксиро-

ванного значения эффективности) определяет масштабируемость параллельного алгоритма для заданной архитектуры. Например, если требуется, чтобы величина W росла экспоненциально, то при такой архитектуре создать параллельную вычислительную систему с необходимым числом процессоров может представлять сложность. Если же требуется, чтобы величина W росла только линейно по отношению к q, то создание подобной вычислительно системы представляется вполне возможным. Будем называть функцию $f(q)$ *функцией изоэффективности*, если W должно расти как $f(q)$ для поддержания постоянной эффективности, а график этой функции – *кривой изоэффективности*.

В [93] приведены кривые изоэффективности некоторых параллельных алгоритмов для архитектур с распределенной и разделяемой памятью (кольцо, гиперкуб). Далее в этой работе мы проведем исследование эффективности параллельного псевдополиномиального алгоритма с поиском в глубину и приведем результаты вычислительных экспериментов с использованием другой архитектуры – вычислительный кластер.

2.6.2.1. Определение эффективности параллельного псевдополиномиального алгоритма с поиском в глубину

Опишем работу псевдополиномиального алгоритма поиска в глубину с использованием параллельного вычислительного кластера из q+1 процессоров. Будем использовать один из процессоров кластера в качестве управляющего, а в качестве стратегии распределения работ использовать метод выделяемых вершин. Управляющий процессор строит дерево решения до уровня $n_0 = \lceil \log_m q \rceil$ (наименьшее целое, большее или равное $\log_m q$), т.е. производит назначение первых n_0 работ на процессоры. При этом количество концевых вершин дерева решений X (соответствующих возможным расписаниям распределенных работ) больше или равно q. Будем предполагать следующее.

- Порядок выбора первых работ не важен и не влияет на получение окончательного решения.

- В случае, если $X > q$, из полученных концевых вершин выбираются q наиболее перспективных для дальнейшего решения. Способ определения перспективности следующий. Для каждой из X вершин определяется задача упорядочения оставшихся $n–n_0$ работ по m процессорам. Для каждой из задач определяется оценка снизу и верхняя оценка длины расписания. Нижняя оценка вычисляется путем решения релаксационной задачи линейного программирования (эта оценка заведомо не хуже аналитической, вычисляемой по формуле $\underline{B} = \max\left\{\max_i\left(\min_j \tau_{ij}\right);\ \frac{\sum_i \min_j \tau_{ij}}{m}\right\}$). Верхняя оценка определяется как меньшее из двух решений, найденных быстрыми эвристическими алгоритмами ПРОП и ВА.

- Вершины, соответствующие решениям с минимальными разностями верхней и нижней оценок, являются более перспективными. Этот факт связан с тем, что при использовании итерационного псевдополиномиального алгоритма с поиском в глубину, чем ближе границы поиска, тем быстрее работает алгоритм.

- Значение рекорда (длины текущего лучшего расписания) есть меньшее из полученных верхних оценок. Вершины, которые соответствуют расписаниям с нижней оценкой, большей рекорда, исключаются из множества активных (далее не рассматриваются).

- В случае, если мощность множества активных вершин меньше q, проводится дополнительное ветвление до уровня $n_0 + 1$, вычисляется новый рекорд и снова строится множества активных вершин.

Полученные q наиболее перспективных активных вершин (или все полученные, если $X=q$) выделяются управляющим процессором рабочим процессорам для дальнейшего построения поддеревьев. Рабочие процессоры строят решение задач с использованием псевдополиномиального алгоритма с поиском в глубину. При получении решения одним из рабочих процессов посылается сигнал завершения с уточнением рекорда (если таковое имело место).

Остальные процессы обновляют информацию о рекорде. В случае, если не все активные вершины были выделены рабочим процессорам для построения поддеревьев, параллельно с работой рабочих

процессоров управляющий процессор проводит дальнейшее ветвление оставшихся активных вершин (и выделение в работу по завершению какого-либо из рабочих процессов) с тем, чтобы по возможности иметь все время q активных вершин. Выдача вершин для построения поддеревьев имеет приоритет над дальнейшим ветвлением, поэтому в итоге все активные вершины будут выданы в работу и обработаны. Таким образом, процесс имеет завершение.

Управляющий процессор также отвечает за хранение информации о текущем рекорде и соответствующем расписании. Поэтому по окончании работы всех рабочих процессоров, управляющий процессор может возвратить информацию об оптимальном расписании.

Определим характер функции изоэффективности для описанного алгоритма. Имеют место следующие соотношения:

$T_{\text{вз}} = U_{\text{вз}} \times N_{\text{вз}}$, где $N_{\text{вз}}$ – число взаимодействий между управляющим и рабочими процессорами;

$$N_{\text{вз}} = 2 \times (q + m^{n-n_0-\alpha}(m^{n_0} - q)).$$

В последней формуле коэффициент 2 введен по той причине, что рабочие процессоры взаимодействуют с управляющим для получения новой работы и обновления рекорда. Величина в скобках отражает количество вершин, отданных рабочим процессам для проведения параллельного построения дерева. Константа α – это эмпирический параметр параллельной вычислительной системы, который отражает глубину дерева, дальше которой управляющий процессор не проводит ветвления. Введение этого параметра связано с тем, что выдача поддерева малой размерности приведет к слишком быстрому завершению ветвления рабочим процессором, который затем будет простаивать в ожидании новой работы. Используя грубую оценку, можно положить, что

$$N_{\text{вз}} = 2 \times (q + m^{n-n_0-\alpha}(m^{n_0} - q)) \approx 2 \times (q + (W+1)\frac{m^{n_0} - q}{m^{n_0+\alpha+1}}).$$

Для времени проведения вычислений (в предположении, что время создания первоначального дерева решения управляющим процессом мало) справедлива оценка

$$T_{\text{выч}} \le U_{\text{выч}} \times q \times m^{n-n_0} = U_{\text{выч}} \times m^n \approx U_{\text{выч}} \times W.$$

Из этого следует формула для E:

$$E = \frac{1}{1 + \frac{2 \times (q + (W+1)\frac{m^{n_0} - q}{m^{n_0+\alpha+1}})U_{\text{вз}}}{U_{\text{выч}} \times W}} = \frac{U_{\text{выч}} \times W}{U_{\text{выч}} \times W + 2 \times (q + (W+1)\frac{m^{n_0} - q}{m^{n_0+\alpha+1}})U_{\text{вз}}}.$$

Для фиксированного значения эффективности E получаем

$$W = -\frac{2EU_{\text{вз}}(qm^{n_0+\alpha+1} + m^{n_0} - q)}{U_{\text{выч}} m^{n_0+\alpha+1}(E-1) + 2EU_{\text{вз}}(m^{n_0} - q)}.$$

В специфике алгоритма заложено то, что количество первоначальных концевых вершин, построенных управляющим процессором, должно быть приблизительно равно числу рабочих процессов, поэтому $m^{n_0} \approx q$. Из этого следуют, что

$$W \approx \frac{qm^{n_0+\alpha+1}E}{(1-E)\frac{U_{\text{выч}}}{U_{\text{вз}}}m^{n_0+\alpha+1}} = \frac{E}{1-E} q \frac{U_{\text{вз}}}{U_{\text{выч}}}.$$

Таким образом, W линейно зависит от q. На рис. 6 изображен характер кривых изоэффективности для различных значений эффективности параллельной вычислительной системы.

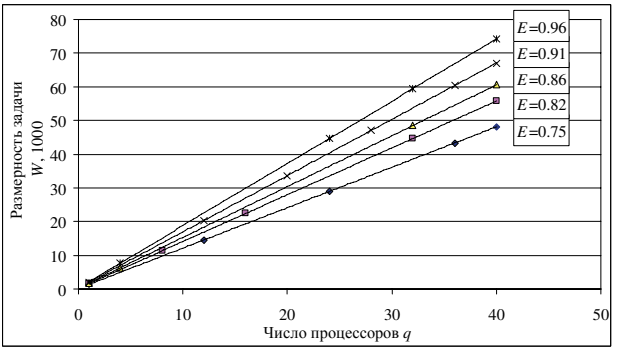


Рис. 6. Кривые изоэффективности

2.7. Результаты экспериментов

2.7.1. Экспериментальная система

Продемонстрируем результаты работы описанных выше алгоритмов. Результаты были получены с использованием персонального компьютера на базе процессора AMD Athlon XP 2800+. Для реализации алгоритмов было разработано программное обеспечение с использованием языков программирования Java и C++. Используемые обозначения: B_1 – длина полученного расписания;

$$\underline{B} = \max\left\{ \max_i \left(\min_j \tau_{ij} \right);\ \frac{\sum_i \min_j \tau_{ij}}{m} \right\} -$$

нижняя оценка времени выполнения множества работ T, получаемая как наибольшее из двух значений: длительности наиболее ресурсоемкой работы, считая, что она распределена на наиболее эффективный для ее выполнения процессор, и времени завершения выполнения всех работ в случае, когда они равномерно распределены на наиболее эффективные для их выполнения процессоры; $\Delta = \frac{B_1 - \underline{B}}{\underline{B}} \times 100\%$ – верхняя оценка относительной погрешности (в процентах); t – время работы алгоритма в секундах; m – количество процессоров; n – количество работ. Для каждого набора (n, m) проводилось 50 экспериментов с произвольными значениями длительностей работ, полученных с помощью программного генератора случайных чисел, позволяющего получать псевдослучайные числа с равномерным распределением на отрезке [1, 1000]. Полученные значения погрешности и времени работы алгоритма затем усреднялись путем отбрасывания 5 самых лучших и 5 самых худших значений и вычисления среднего арифметического оставшихся значений.

2.7.2. Таблицы и графики

В табл. 1 приведены результаты для задач, времена в которых задаются формулой $\tau_i = n - i + 1,\ i = \overline{1,n}$.

Таблица 1. *Арифметическая прогрессия*

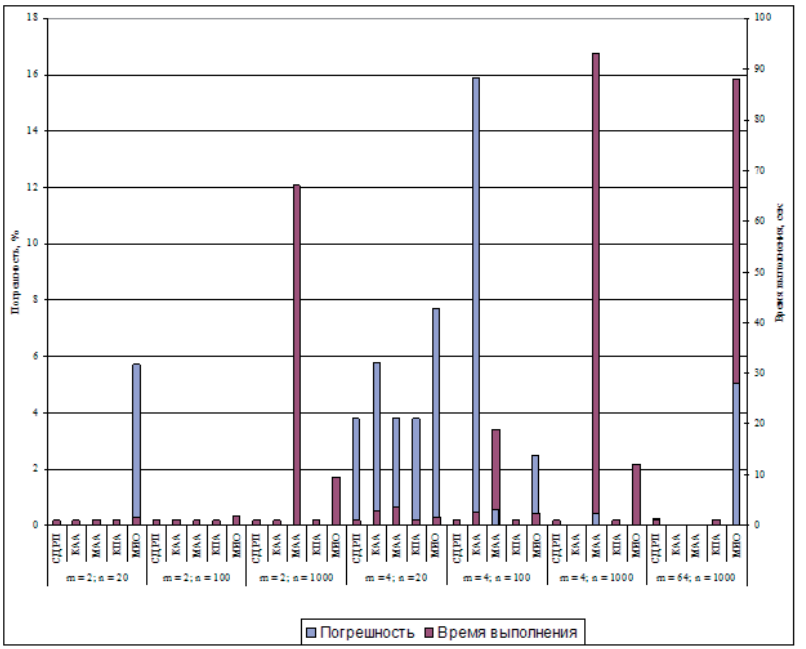
Задача	Алгоритм	Параметр алгоритма	$\underline{B}$	B_1	Δ, %	t, сек.
$m = 2$; $n = 20$	СДРП		105	105	0	<1
	СКРП			110	4.76	<1
	ПАА	$s = 2$		105	0	<1
	КАА	$s = 2$		105	0	<1
	МАА	$s = 2$		105	0	<1
	точный			105	0	5.0
	КПА	$s = 1$		105	0	<1
	МИО			111	5.71	1.5
$m = 2$; $n = 100$	СДРП		2525	2525	0	<1
	СКРП			2550	1	<1
	ПАА	$s = 10$		2525	0	1.5
	КАА	$s = 10$		2525	0	1.0
	МАА	$s = 10$		2525	0	<1
	КПА	$s = 1$		2525	0	<1
	МИО			2530	0.2	2.0
$m = 2$; $n = 1000$	СДРП		250250	250250	0	<1
	СКРП			250500	0.1	<1
	КАА	$s = 50$		250300	0.02	<1
	МАА	$s = 48$		250250	0	67
	КПА	$s = 1$		250250	0	<1
	МИО			250252	~0	9.5
$m = 4$; $n = 20$	СДРП		52	54	3.8	<1
	СКРП			60	15.4	<1
	ПАА	$s = 2$		54	3.8	3.2
	КАА	$s = 2$		55	5.8	3.1

	МАА	$s = 2$		54	3.8	3.8
	Точный			52	0	10 сут.
	КПА	$s = 2$		54	3.8	<1
	МИО			56	7.7	1.5
$m = 4$; $n = 100$	СДРП		1262	1264	0.2	<1
	СКРП			1300	3	<1
	КАА	$s = 10$		1463	15.9	2.7
	МАА	$s = 10$		1269	0.55	19
	КПА	$s = 2$		1263	0.1	<1
	МИО			1293	2.5	2.4
$m = 4$; $n = 1000$	СДРП		125125	125125	0	<1
	СКРП			125500	0.3	<1
	МАА	$s = 166$		125500	0.3	93
	КПА	$s = 2$		125125	0	<1
	МИО			125290	0.1	12.1
$m = 64$; $n = 1000$	СДРП		7820	7838	0.23	<1
	СКРП			8320	6.4	<1
	КПА	$s = 16$		7837	0.21	<1
	МИО			8218	5.1	88

Для удобства наиболее показательные результаты из табл. 1 сведены в диаграмму 1. Как видно из приведенных выше результатов, все агрегирующие алгоритмы дают результаты по погрешности лучше, чем метод имитации отжига для данного распределения длительностей работ. По времени работы на многих входах МИО также уступает агрегирующим алгоритмам. Среди агрегирующих алгоритмов худшие результаты по погрешности дает КАА, что ожидаемо, т.к. решение агрегированной задачи проводится с помощью быстрой эвристики. Наиболее перспективными представляются алгоритмы МАА и КПА. Последний представляет собой на самом деле метод улучшения оценки расписания, полученного первоначальным быстрым алгоритмом. Для тех случаев, когда изначальное приближение близко к оптимуму (как, например, при данном распределении), КПА за приемле-

мое время находит улучшение расписания. Время работы алгоритма МАА наиболее подвержено влиянию размерности входа, однако, в большинстве случаев найденное им решение было лучшим.

Диаграмма 1. Результаты для длительностей работ, заданных арифметической прогрессией



В табл. 2 приведены результаты для задач, времена в которых задаются выражением $\tau_i = n - i + 1 + [1.2^{n-i+1}],\ i = \overline{1,n}$.

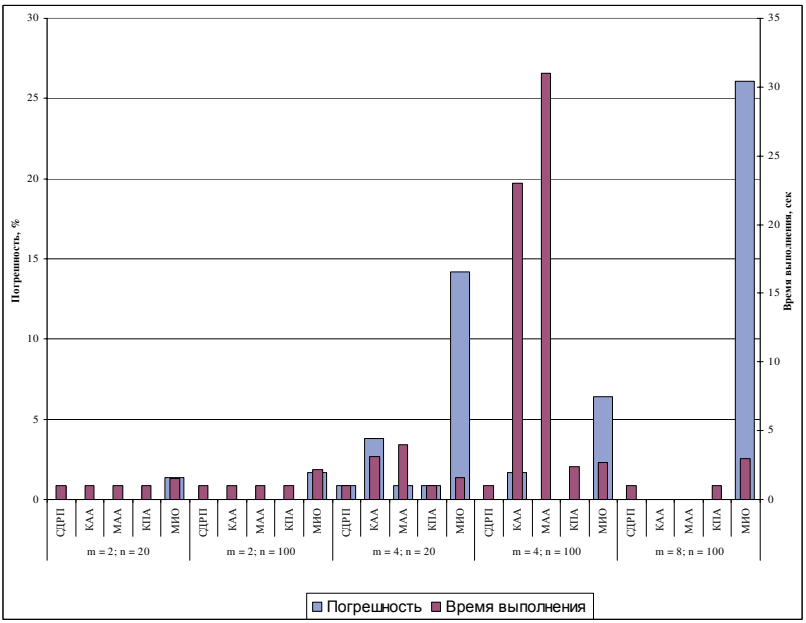
Таблица 2. Распределение, близкое к геометрической прогрессии

Задача	Алгоритм	Параметр алгоритма	$\underline{B}$	B_1	Δ, %	t, сек.
$m = 2$; $n = 20$	СДРП		212	212	0	<1
	СКРП			228	7,5	<1
	ПАА	$s = 2$		212	0	<1

	КАА	$s = 2$		212	0	<1
	МАА	$s = 2$		212	0	<1
	КПА	$s = 1$		212	0	<1
	МИО			215	1,4	1,5
$m = 2$; $n = 100$	СДРП		248456416	248456419	0	<1
	СКРП			225869676	9,1	<1
	ПАА	$s = 10$		207047431	0	3,0
	КАА	$s = 10$		207145697	0	<1
	МАА	$s = 10$		248456419	0	<1
	КПА	$s = 1$		248456419	0	<1
	МИО			252646578	1,7	2,2
$m = 4$; $n = 20$	СДРП		106	107	0,9	<1
	СКРП			127	19,8	<1
	ПАА	$s = 2$		107	0,9	3,3
	КАА	$s = 2$		110	3,8	3,1
	МАА	$s = 2$		107	0,9	4,0
	КПА	$s = 2$		107	0,9	<1
	МИО			121	14,2	1,6
$m = 4$; $n = 100$	СДРП		124228208	124228210	0	<1
	СКРП			159012106	28	<1
	КАА	$s = 10$		105279930	1,7	23
	МАА	$s = 10$		124228210	0	31
	КПА	$s = 2$		124228210	0	2,4
	МИО			132145694	6,4	2,7
$m = 8$; $n = 100$	СДРП		82818074	82818074	0	<1
	СКРП			89930424	30,3	<1
	КПА	$s = 4$		82818074	0	<1
	МИО			104461884	26,1	3,0
$m = 16$; $n = 100$	СДРП		82818074	82818074	0	<1
	СКРП			112632580	36	<1
	КПА	$s = 4$		82818074	0	<1
	МИО			82818074	0	4,5

Для простоты сравнения сведем результаты в диаграмму 2 (последний вариант не внесен в связи с тем, что длина расписания ограничена длительностью самой длинной работы, и это расписание было найдено всеми алгоритмами, за исключением СКРП).

Диаграмма 2. Результаты для длительностей работ, заданных геометрической прогрессией



Из приведенных выше результатов видно, что при существенном различии длительностей работ, наблюдаемая картина остается прежней – метод имитации отжига дает результаты хуже любого агрегирующего алгоритма, алгоритмы МАА и КПА являются наиболее перспективными.

В табл. 3 – 9 приведены статистические результаты работы жадных эвристик, агрегирующих алгоритмов и, для сравнения, метода имитации отжига для задачи упорядочения с идентичными процессорами. Эти результаты также сведены в диаграмму 3 для удобства сравнительного анализа.

Таблица 3. Статистические результаты работы ПАА

Условия задачи	$\Delta_{ср.}$, %	$t_{ср.}$, сек.
$M = 2$; $n = 20$; $s = 2$	0.01	<1
$M = 2$; $n = 100$; $s = 10$	~0	2.0
$M = 4$; $n = 20$; $s = 2$	3	2.3

Таблица 4. Статистические результаты работы КАА

Условия задачи	$\Delta_{ср.}$, %	$t_{ср.}$, сек.
$M = 2$; $n = 20$; $s = 2$	0.1	<1
$M = 2$; $n = 100$; $s = 10$	0.01	<1
$M = 4$; $n = 20$; $s = 2$	2	2.4
$M = 4$; $n = 100$; $s = 10$	4.5	10
$M = 2$; $n = 1000$; $s = 100$	0,7	<1
$M = 4$; $n = 1000$; $s = 100$	2.5	37

Таблица 5. Статистические результаты работы МАА

Условия задачи	$\Delta_{ср.}$, %	$t_{ср.}$, сек.
$m = 2$; $n = 20$; $s = 2$	0.05	<1
$m = 2$; $n = 100$; $s = 8$	~0	1
$m = 4$; $n = 20$; $s = 2$	0.5	2.1
$m = 4$; $n = 100$; $s = 16$	0.5	12
$m = 2$; $n = 1000$; $s = 166$	1.4	2.2
$m = 2$; $n = 1000$; $s = 50$	0.2	110
$m = 4$; $n = 384$; $s = 48$	0,02	80
$m = 4$; $n = 1000$; $s = 100$	2.4	317

Таблица 6. Статистические результаты работы СДРП

Условия задачи	$\Delta_{ср.}$, %	$t_{ср.}$, миллисек.
$m = 2$; $n = 20$	0.5	<10
$m = 2$; $n = 100$	0.1	<10
$m = 4$; $n = 20$	5	<10
$m = 4$; $n = 100$	1	10
$m = 8$; $n = 100$	7	10

$m = 16; n = 100$	1	10
$m = 2; n = 1000$	2.5	20
$m = 4; n = 1000$	3.5	20
$m = 4; n = 10000$ [1]	4.6	500

Таблица 7. Статистические результаты работы СКРП

Условия задачи	$\Delta_{ср.}$, %	$t_{ср.}$, миллисек.
$m = 2; n = 20$	5	<10
$m = 2; n = 100$	2	<10
$m = 4; n = 20$	15	<10
$m = 4; n = 100$	3	10
$m = 8; n = 100$	7	10
$m = 16; n = 100$	15	10
$m = 2; n = 1000$	5	20
$m = 4; n = 1000$	5	20
$m = 4; n = 10000$	60	500

Таблица 8. Статистические результаты работы КПА

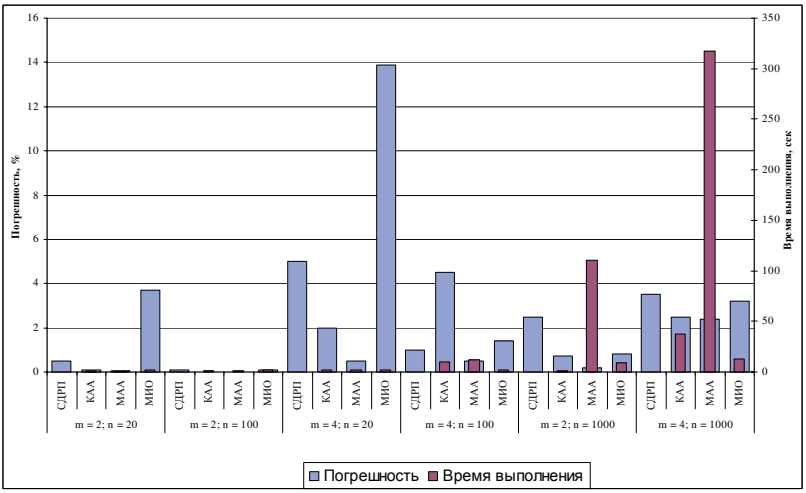
Условия задачи	$\Delta_{ср}$, %	$t_{ср.}$, сек.
$m = 4; n = 60; s = 2$	8.6	23
$m = 4; n = 80; s = 2$	8.7	27
$m = 4; n = 100; s = 2$	10.2	37
$m = 6; n = 100; s = 3$	7.5	40

Таблица 9. Статистические результаты работы МИО

Условия задачи	$\Delta_{ср.}$, %	$t_{ср.}$, сек.
$m = 2; n = 20$	3.7	1.5
$m = 2; n = 100$	0.1	2.1
$m = 4; n = 20$	13.9	1.6
$m = 4; n = 100$	1.4	2.4
$m = 2; n = 1000$	0,8	9.1
$m = 4; n = 1000$	3.2	12.4

[1] Времена работ задавались в границах от 10 до 10000.

Диаграмма 3. Статистические результаты



Из приведенных выше результатов следует, что для задачи составления расписания работ на идентичных процессорах метод агрегирования показывает хорошие результаты – при достаточно малом времени работы длина расписания близка к нижней оценке.

Из разработанных алгоритмов агрегирования особенно необходимо отметить многоуровневый агрегирующий алгоритм – на большинстве входах, полученное решение лучше, чем найденные с помощью жадных эвристик и метода имитации отжига; при этом время работы алгоритма достаточно мало, чтобы позволить использовать его при решении многих практических задач.

В табл. 10 приведены результаты работы параллельной реализации псевдополиномиального алгоритма с поиском в глубину на параллельном вычислительном кластере для задачи составления расписания на произвольных процессорах. Из приведенных результатов видно, что в задачах, где необходимо найти точное решение, использование параллельной реализации ПАПГ оправдано. Алгоритм обладает высокой эффективностью распараллеливания и масштабируемостью, что позволяет надеяться на возможность его использования при решении многих практических задач.

Таблица 10. Результаты параллельного процесса упорядочения с использованием кластера

Условия задачи	Время работы, T_q, сек	Прирост в скорости	Эффективность
m=2; n=30; q=2	3.5	1.48	0.74
m=2; n=40; q=2	42	1.79	0.89
m=4; n=24; q=2	1.3	1.15	0.58
m=4; n=30; q=2	21	1.90	0.95
m=4; n=30; q=4	10.6	3.77	0.94
m=8; n=30; q=2	60.3	1.99	0.995
m=8; n=30; q=4	30.2	3.94	0.993

В табл. 11 – 14 приведены результаты алгоритмов, разработанных в данной главе для задачи составления расписания на произвольных процессорах, и для сравнения даны результаты работы метода имитации отжига. Для удобства результаты сведены в диаграммах 4 и 5.

Таблица 11. Результаты работы алгоритма ПРОП

Условия задачи	$\Delta_{ср.}$, %	$t_{ср.}$, мс
m = 2; n = 20	18.2	1
m = 2; n = 40	14.6	1
m = 2; n = 50	14.2	2
m = 4; n = 20	24.8	1
m = 4; n = 100	16.6	4
m = 8; n = 30	47.6	1
m = 16; n = 100	36.7	4
m = 2; n = 1000	13.6	16
m = 4; n = 5000	25.1	42
m = 4; n = 10000	12.6	200

Таблица 12. Результаты работы алгоритма ВА

Условия задачи	$\Delta_{ср.}$, %	$t_{ср.}$, сек.
m = 2; n = 20	6.1	0.1

$m = 2$; $n = 40$	3.2	0.2
$m = 2$; $n = 50$	3.1	0.2
$m = 4$; $n = 100$	6	0.2
$m = 8$; $n = 30$	46.3	0.1
$m = 4$; $n = 1000$	0.6	1.2
$m = 8$; $n = 1000$	1.2	3.1
$m = 16$; $n = 500$	10.7	2.44
$m = 16$; $n = 500$;	9.1	2.27
$m = 16$; $n = 500$;	9.9	2.24
$m = 16$; $n = 1000$;	5.3	8.7
$m = 2$; $n = 1000$;	0.2	1.0
$m = 2$; $n = 5000$;	0.02	23
$m = 4$; $n = 5000$;	0.1	56

Таблица 13. Результаты работы алгоритма ПАПГ

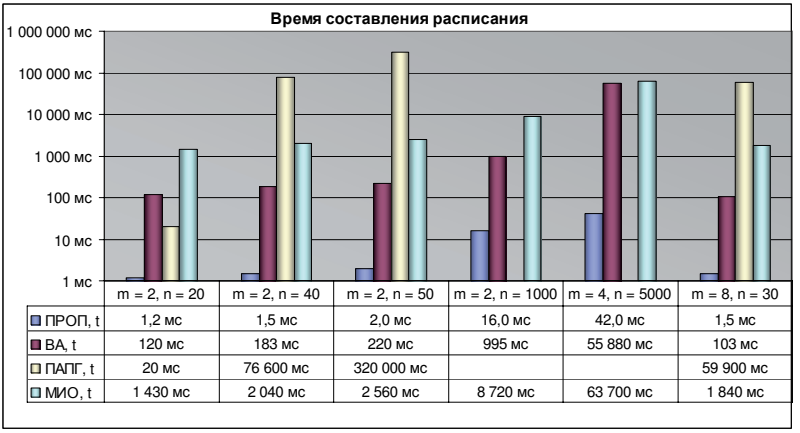
Условия задачи	$t_{ср.}$, сек.
$m = 2$; $n = 20$	0.02
$m = 2$; $n = 24$;	0.36
$m = 2$; $n = 30$;	5.2
$m = 2$; $n = 40$;	76
$m = 2$; $n = 50$; $\varepsilon = 2.9\%$	320
$m = 4$; $n = 20$;	0.1
$m = 4$; $n = 24$;	1.5
$m = 4$; $n = 30$;	40
$m = 8$; $n = 24$;	3.5
$m = 8$; $n = 30$;	120

Таблица 14. Результаты работы алгоритма МИО

Условия задачи	$\Delta_{ср.}$, %	$t_{ср.}$, сек.
$m = 2$; $n = 20$	12.6	1.4
$m = 2$; $n = 40$	13.7	2.0
$m = 2$; $n = 50$	16.4	2.5
$m = 2$; $n = 1000$	12.3	8.7

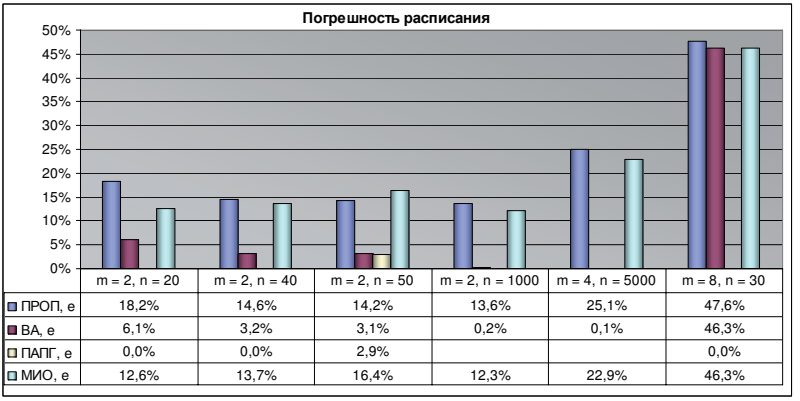
$m = 4$; $n = 5000$	22.9	63.7
$m = 8$; $n = 30$;	46.3	1.8

Диаграмма 4. Время работы алгоритмов составления расписания для произвольных процессоров



	m = 2, n = 20	m = 2, n = 40	m = 2, n = 50	m = 2, n = 1000	m = 4, n = 5000	m = 8, n = 30
ПРОП, t	1,2 мс	1,5 мс	2,0 мс	16,0 мс	42,0 мс	1,5 мс
ВА, t	120 мс	183 мс	220 мс	995 мс	55 880 мс	103 мс
ПАПГ, t	20 мс	76 600 мс	320 000 мс			59 900 мс
МИО, t	1 430 мс	2 040 мс	2 560 мс	8 720 мс	63 700 мс	1 840 мс

Диаграмма 5. Погрешность алгоритмов составления расписания для произвольных процессоров



	m = 2, n = 20	m = 2, n = 40	m = 2, n = 50	m = 2, n = 1000	m = 4, n = 5000	m = 8, n = 30
ПРОП, e	18,2%	14,6%	14,2%	13,6%	25,1%	47,6%
ВА, e	6,1%	3,2%	3,1%	0,2%	0,1%	46,3%
ПАПГ, e	0,0%	0,0%	2,9%			0,0%
МИО, e	12,6%	13,7%	16,4%	12,3%	22,9%	46,3%

Приведенные результаты показывают, что среди эвристических алгоритмов наиболее перспективным является ВА: при достаточно малой погрешности алгоритм обладает и малым временем работы. Два других эвристических алгоритма: ПРОП и МИО получают решение примерно с одинаковым качеством, однако, ПРОП находит реше-

ние существенно быстрее. Все эвристики обладают особенностью, что с ростом отношения m/n растет и погрешность получаемого решения.

Алгоритмы ВА и ПАПГ могут использоваться в виде пакета прикладных алгоритмов для решения практических задач: на входах со сравнительно малым числом задач (а следовательно б**о**льшим отношением m/n) ПАПГ достаточно быстро улучшает решение, полученное ВА до оптимума или до решения с гарантированной заданной точностью. При решении задач с большим числом работ, когда получение точного решения невозможно, использование ВА оправдано за счет малой погрешности получаемого расписания.

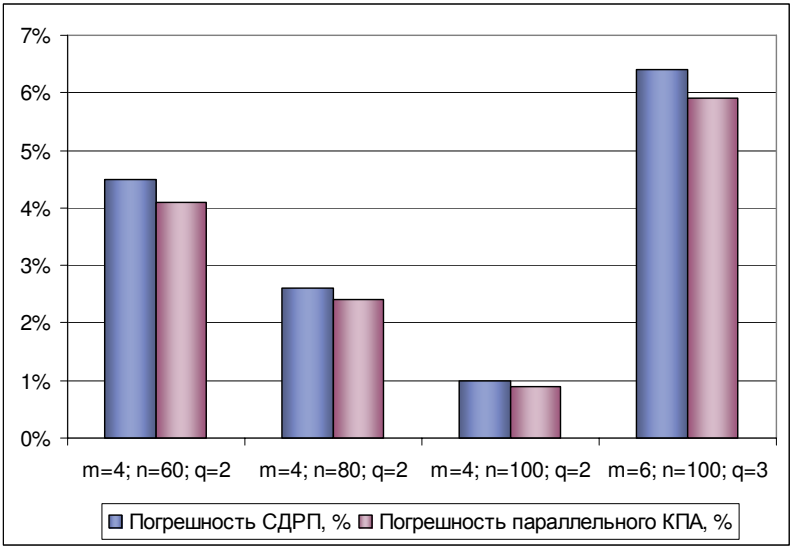
В табл. 15 приведены результаты выполнения КПА на параллельном вычислительном кластере. Алгоритм демонстрирует заметное улучшение первоначальной оценки (порядка 10%) и высокую эффективность распараллеливания.

Таблица 15. Результаты параллельной работы КПА

Задача	Погрешность		Улучшение расписания, %	Время работы КПА, сек.	Прирост в скорости
	СДРП, %	Параллельного КПА, %			
m=4; n=60; $\tau_i \in$ [1,1000]; q=2	4.5	4.1	8.6	12	1.94
m=4; n=80; $\tau_i \in$ [1,10000]; q=2	2.6	2.4	8.7	13.8	1.96
m=4; n=100; $\tau_i \in$ [1,10000]; q=2	1	0.9	10.2	18.8	1.97
m=6; n=100; $\tau_i \in$ [1,10000]; q=3	6.4	5.9	7.5	13.4	2.98

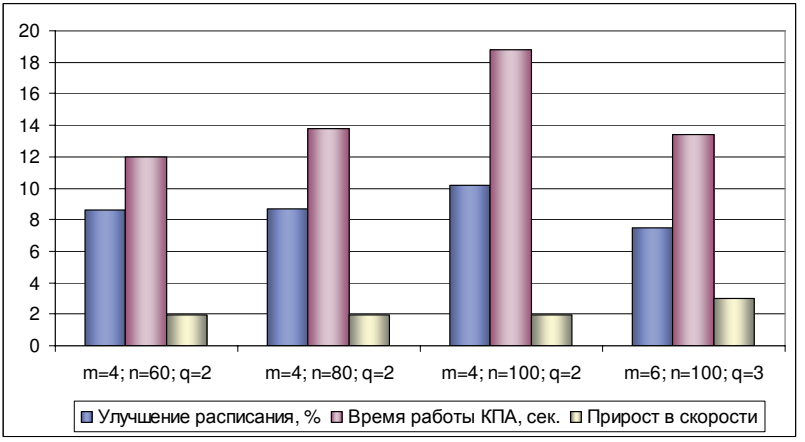
На диаграмме 6 для удобства сравнительного анализа приведены результаты погрешности алгоритмов СДРП и параллельной реализации КПА.

Диаграмма 6. Сравнительный анализ погрешности алгоритмов СДРП и параллельного КПА



На диаграмме 7 для удобства анализа приведены результаты работы параллельной реализации КПА.

Диаграмма 7. Результаты параллельной реализации КПА



Глава 3.
Некоторые задачи построения многопроцессорных расписаний с дополнительными ограничениями

В настоящей главе рассмотрены задачи (1) составления допустимого расписания с прерываниями в многопроцессорной системе в случае, когда заданы директивные интервалы, а длительности выполнения работ линейно зависят от количества выделенного им дополнительного ресурса; (2) построения допустимых многопроцессорных расписаний с прерываниями в случае, когда требования на выполнение работ поступают циклически с заданными периодами; (3) составления многопроцессорных расписаний без прерываний при наличии неопределенных факторов.

3.1. Алгоритмы составления многопроцессорных расписаний с прерываниями в системах с нефиксированными длительностями работ

В настоящем разделе рассмотрена задача составления допустимого расписания с прерываниями в многопроцессорной системе в случае, когда заданы директивные интервалы, а длительности выполнения работ линейно зависят от количества выделенного им дополнительного ресурса. Разработан алгоритм, основанный на сведении исходной задачи к задаче линейного программирования в случае одинаковых директивных интервалов и к задаче о потоке минимальной стоимости для произвольных директивных интервалов.

Подобная задача для случая, когда дополнительный ресурс отсутствует, рассматривалась в [1] и была сведена к задаче о максимальном потоке в сети специального вида. Задача с произвольными процессорами и произвольными директивными интервалами рассматривалась в [11], а задача с произвольными процессорами и одинаковыми директивными интервалами – в [12]. В обеих этих работах дополнительный ресурс не рассматривался. Задача минимизации времени выполнения сетевого комплекса работ, когда длительности выполнения работ являются функциями от вектора распределения ресурсов,

а число процессоров не ограничено, рассматривалась в [111] и была сведена к задаче нелинейного программирования.

3.1.1. Постановка задачи

Рассматривается вычислительная система, состоящая из m идентичных процессоров, и множество работ $N=\{1, 2, \ldots, n\}$. Предполагается, что в каждый момент времени каждый процессор может выполнять не более одной работы, а каждая работа выполняется не более чем одним процессором. При выполнении работ допускаются прерывания и переключения с одного процессора на другой. Предполагается, что прерывания и переключения не сопряжены с временными затратами. Для работы $i \in N$ установлен директивный интервал $(b_i, f_i]$ (т.е. работа i может выполняться только в этом интервале). Помимо процессоров в системе имеется дополнительный ресурс не возобновляемого типа. Суммарное количество этого ресурса составляет R. Если работе i выделено r_i единиц дополнительного ресурса, то ее длительность составляет $t_i = d_i - a_i r_i$, где

$$r_i \in [0, \quad \bar{r}_i], \; i = 1, 2, \ldots, n, \tag{1}$$

$$\sum_{i \in N} r_i \le R, \tag{2}$$

a_i, d_i, $\bar{r}_i$ – заданные величины, $a_i > 0$, $d_i > 0$, $0 \le \bar{r}_i < d_i / a_i$. Таким образом, $t_i \in [d_i - a_i \bar{r}_i, d_i]$, причем $d_i - a_i \bar{r}_i > 0$. Требуется найти такое распределение ресурсов $(r_1^0, r_2^0, \ldots, r_n^0)$, при котором существует допустимое расписание (т.е. такое расписание, при котором каждая работа выполняется в своем директивном интервале), или установить, что такого распределения ресурсов не существует. При этом распределение ресурсов $(r_1^0, r_2^0, \ldots, r_n^0)$ должно удовлетворять ограничениям (1), (2). Кроме того, требуется найти минимальную величину R дополнительного ресурса, при которой допустимое расписание существует.

3.1.2. Задача с одинаковыми директивными интервалами

В этом разделе предполагается, что директивные интервалы всех работ совпадают. Без ограничения общности можно считать, что $b_i = 0$, $f_i = F$ ($i \in N$). Как следует из [1], необходимым и достаточным условием существования допустимого расписания в этом случае является выполнение неравенства $\sum_{i \in N} t_i \le mF$, или $\sum_{i \in N} (d_i - a_i r_i) \le mF$, $\sum_{i \in N} a_i r_i \le \sum_{i \in N} d_i - mF$. Пусть $\sum_{i \in N} d_i - mF = B$. Тогда задача заключается в поиске такого распределения ресурсов ($r_1, r_2, \ldots, r_n$), которое удовлетворяет системе ограничений

$$\sum_{i \in N} a_i r_i \ge B,$$
$$\sum_{i \in N} r_i \le R, \qquad (3)$$
$$r_i \in [0, \bar{r}_i], i = 1, 2, \ldots, n.$$

Таким образом, допустимое расписание существует тогда и только тогда, когда задача линейного программирования (3) имеет решение. Если задача (3) имеет решение $(r_1^0, r_2^0, \ldots, r_n^0)$, то определив $t_i^0 = d_i - a_i^0$ $(i \in N)$ и применив алгоритм упаковки [1], найдем допустимое расписание. Вычислительная сложность алгоритма Кармаркара для решения задачи линейного программирования $O(n^{4,5} \log_2(nT))$, а алгоритма упаковки – $O(n)$.

Определим минимальное количество R_{min} дополнительного ресурса, при котором допустимое расписание существует. Опишем первый метод решения этой задачи. Рассмотрим задачу линейного программирования

$$R \to min$$
$$\sum_{i \in N} a_i r_i \ge B,$$
$$\sum_{i \in N} r_i \le R, \qquad (4)$$
$$r_i \in [0, \bar{r}_i], i = 1, 2, \ldots, n,$$
$$R \ge 0.$$

Задача (4) имеет решение. Каждому ее решению $(r_1^0, r_2^0, ..., r_n^0)$, R_{min} соответствует допустимое расписание, а R_{min} – минимально допустимое количество дополнительного ресурса.

Опишем второй метод поиска величины $R_{\min}$ Перепишем систему (3) следующим образом:

$$\sum_{i \in N} a_i r_i \geq B,$$

$$\sum_{i \in N} r_i \leq R + R', \qquad (5)$$

$$r_i \in [0, \bar{r}_i], \; i = 1, 2, \ldots, n.$$

Требуется определить такую величину R', для которой система (5) имеет решение. В этом случае можно построить допустимое расписание. Величину R' будем определять методом деления отрезка пополам. Алгоритм останавливается, когда длина получаемого отрезка не превосходит заданной величины Δ. Приведем описание алгоритма.

Шаг 1. Определить интервал поиска величины $R' \in [R'_{10}, R'_{20}]$. При $R' = R'_{10}$ решение системы (5) не существует, а при $R' = R'_{20}$ существует. Положить $R_1 = R'_{10}, \quad R_2 = R'_{20}$.

Шаг 2. Положить $R' = (R_1 - R_2)/2$. Если система (5) имеет решение, то перейти на шаг 3, в противном случае – на шаг 4.

Шаг 3. Положить $R_2 = R'$. Перейти на шаг 5.

Шаг4. Положить $R_1 = R'$. Перейти на шаг 5.

Шаг 5. Если $R_2 - R_1 \geq \Delta$, то о перейти на шаг 2, в противном случае – на шаг 6.

Шаг 6. Работа алгоритма завершена. Величина $R + R'$ является решением задачи поиска дополнительного ресурса, при котором допустимое расписание существует.

Определим начальные границы $R'_{10}, \quad R'_{20}$ диапазона для величины R'. Очевидно, что в качестве R'_{10} можно взять нулевое значение: $R'_{10} = 0$. Ведём вспомогательную задачу линейного программирования:

$$\sum_{i \in N} a_i r_i \to \max,$$

$$\sum_{i\in N} r_i \le R, \tag{6}$$

$$r_i \in [0,\ \bar{r}_i],\ i = 1, 2, \ldots, n.$$

Пусть $\max \sum_{i\in N} a_i r_i = B' \le B$ и пусть этот максимум достигается на векторе $(r_1', r_2', \ldots, r_n')$.

Покажем, что в качестве R'_{20} можно взять величину

$$R' = \frac{(B - B')\sum_{i\in N}(\bar{r}_i - r_i')}{\sum_{i\in N} a_i(\bar{r}_i - r_i') - (B - B')}. \tag{7}$$

Рассмотрим следующий вектор решений:

$$r_i = r_i' + \frac{\bar{r}_i - r'}{\sum_{i\in N}(\bar{r}_i - r_i') + R'} R', i \in N. \tag{8}$$

Нетрудно заметить, что это решение удовлетворяет следующим существенным ограничениям системы (5):

$$\sum_{i\in N} r_i \le R + R',$$

$$r_i \in [0,\ \bar{r}_i],\ i = 1, 2, \ldots, n.$$

Действительно, $r_i \ge 0$ и, кроме того, из (8) следует, что $r_i = r_i' + (\bar{r}_i - r_i')\alpha$, где $\alpha \in (0,1)$. Поэтому $r_i \le \bar{r}$. Далее,

$$\sum_{i\in N} r_i = \sum_{i\in N} r_i' + \frac{R'}{\sum_{i\in N}(\bar{r}_i - r_i') + R'} \sum_{i\in N}(\bar{r}_i - r_i') \le R + R'.$$

Подставим в этот неравенство $\sum_{i\in N} a_i r_i \ge B$ величины r_i, вычисленные по формуле (8):

$$\sum_{i\in N} a_i r_i' + \frac{R'}{\sum_{i\in N}(\bar{r}_i - r_i') + R'} \sum_{i\in N} a_i(\bar{r}_i - r_i') \ge B.$$

С помощью простых преобразований несложно показать, что последнее неравенство эквивалентно неравенству

$$R' \ge \frac{(B - B')\sum_{i\in N}(\bar{r}_i - r_i')}{\sum_{i\in N} a_i(\bar{r}_i - r_i') - (B - B')}.$$

Из последнего неравенства следует справедливость выбора величины R'_{20} по формуле (7).

На рис. 1 приведен график зависимости времени (в условных единицах) работы первого алгоритма решения задачи (4) от числа работ. На рис. 2 приведены времена (в условных единицах) работы первого и второго алгоритмов при различных значениях Δ в зависимости от числа работ (значения Δ указаны в скобках). Из этих результатов видно, что при $\Delta \geq 1$ второй алгоритм работает существенно быстрее первого, при $\Delta \leq 0{,}01$ первый работает быстрее, а при $\Delta = 0{,}1$ времена работы алгоритмов приблизительно одинаковые.

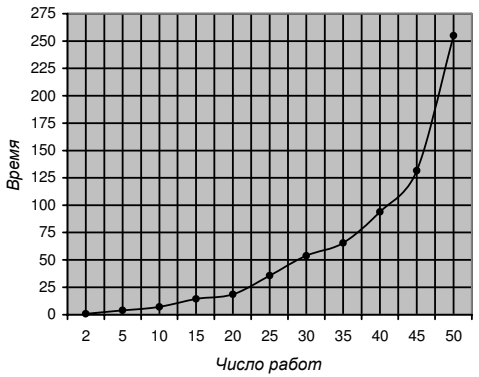


Рис. 1. Зависимость времени выполнения первого алгоритма от числа работ.

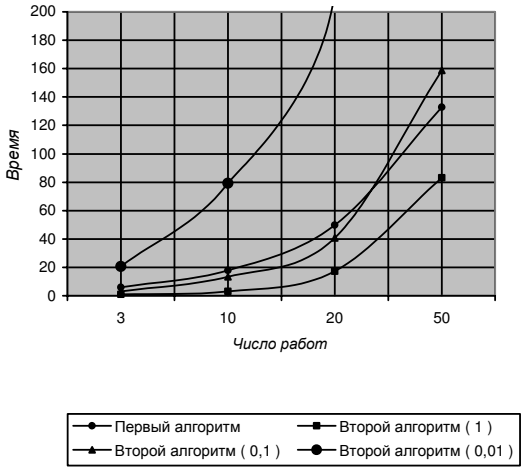


Рис. 2. Зависимость времени выполнения первого и второго алгоритмов от числа работ и величины Δ.

3.1.3. Задача с произвольными директивными интервалами

Теперь будем предполагать, что заданы произвольные директивные интервалы $(b_i, f_i]$. Покажем, что в этом случае исходная задача может быть сведена к задаче о потоке минимальной стоимости. По аналогии с тем, как это сделано в [1], построим потоковую сеть $G = (V, A)$ (рис. 3) с источником s и стоком t (V – множество узлов сети, A – множество дуг).

Пусть $y_0 < y_1 < \ldots < y_p$ $(p < 2n)$ – все различные по величине значения b_i, f_i $(i \in N)$. Определим $V = \{s, t, I_1, I_2, \ldots, I_p, w_1, w_2, \ldots, w_n\}$, где узел I_j соответствует интервалу $(y_{j-1}, y_j]$ $(j = 1, 2, \ldots, p)$, а узел w_i – работе $i \in N$. Множество дуг A сети G определим следующим образом: (s, I_j) $(j = 1, 2, \ldots, p)$; (w_i, t) $(i \in N)$; (I_j, w_i) в случае, если $(y_{j-1}, y_j] \subseteq (b_i, f_i]$ (отметим, что для любых j $(1 \le j \le p)$ и $i \in N$ интервал $(y_{j-1}, y_j]$ либо не пересекается с интервалом $(b_i, f_i]$, либо целиком лежит в нем); определим также возвратную дугу (t, s). Пусть $\Delta_j = y_j - y_{j-1}$ – длина интервала I_j. Для каждой дуги определим три параметра: L, U, C, где L – нижняя граница потока по дуге, U –

верхняя граница потока по дуге, C – стоимость единицы потока по дуге. Параметры дуг сети G указаны в табл. 1.

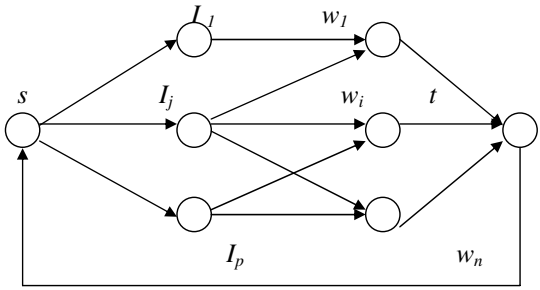


Рис.3. Потоковая сеть G.

Таблица. 1. Параметры дуг сети G.

Дуга	L	U	C
(s, I_j)	0	$m\Delta_j$	0
(I_j, w_i)	0	Δ_j	0
(w_i, t)	$d_i - a_i \bar{r}_i$	d_i	$-1/a_i$
(t, s)	0	$\sum_{i \in N} d_i$	0

Лемма 1. Для существования допустимого расписания необходимо и достаточно существование в сети G циркуляции g, стоимость которой

$$c(g) \le R - \sum_{i \in N} d_i / a_i . \qquad (9)$$

Доказательство. 1) *Необходимость*. Пусть в рассматриваемой задаче существует допустимое расписание и пусть r_i – величина дополнительного ресурса, выделенного работе i, а $t_i = d_i - a_i r_i$ – длительность работы i. Тогда $r_i = (d - t_i) / a_i$ $(i \in N)$, а суммарная величина дополнительного ресурса, необходимая для выполнения всего комплекса работ N, равна $\sum_{i \in N} (d_i - t_i) / a_i \le R$, откуда следует неравенство

$$\sum_{i \in N} (-t_i / a_i) \le R - \sum_{i \in N} d_i / a_i . \qquad (10)$$

По аналогии с тем, как это сделано в [1], можно показать, что в сети G существует такая циркуляция g (т.е. поток, для которого условие сохранения выполнено в каждом узле сети G, включая источник s и сток t), что

$$g_{it} = t_i \qquad (11)$$

(g_{it} – величина потока по дуге (w_i, t)). Стоимость $c(g)$ этой циркуляции равна $c(g) = \sum_{i \in N} (-g_{it} / a_i)$. Из (10), (11) следует, что $c(g) = \sum_{i \in N} (-t_i / a_i) \le R - \sum_{i \in N} d_i / a_i$, т.е. выполнено неравенство (9).

2) *Достаточность*. Пусть в сети G существует циркуляция g, для которой выполнено неравенство (9). Тогда по аналогии с тем, как это сделано в [1], можно показать, что существует допустимое расписание, в котором длительность t_i каждой работы $i \in N$ удовлетворяет равенству (11). Параметры дуг (w_i, t) определены таким образом, что $t_i \in [d_i - a_i \bar{r}_i, d_i]$, а стоимость циркуляции g равна $c(g) = \sum_{i \in N} (-t_i / a_i)$. Из (9) следует, что $\sum_{i \in N} (-t_i / a_i) \le R - \sum_{i \in N} d_i / a_i$ и $\sum_{i \in N} (d_i - t_i) / a_i \le R$. Поскольку $(d_i - t_i) / a_i$ – это величина дополнительного ресурса, выделяемого работе i, то из последнего неравенства следует, что суммарное количество дополнительного ресурса, требуемое для реализации допустимого расписания, не превосходит R. Лемма доказана.

Опишем алгоритм решения поставленной задачи, основанный на доказанной лемме.

Шаг 1. Построить потоковую сеть G.

Шаг 2. Найти циркуляцию g минимальной стоимости в сети G (для этого можно применить, например, алгоритм дефекта [112]). Пусть $c(g)$ – ее стоимость, а g_{it} – величина потока по дуге (w_{it}, t).

Шаг 3. Если $c(g) > R - \sum_{i \in N} d_i / a_i$, то допустимого расписания не существует. Если $c(g) \le R - \sum_{i \in N} d_i / a_i$, то допустимое расписание существует. При этом величина g_{it} потока по дуге (I_j, w_i) равна величине процессорного времени, выделяемого работе i в интервале I_j

($i \in N$; $j = 1, 2, \ldots, p$). Для построения расписания в интервале I_j следует применить алгоритм упаковки, а для построения искомого расписания следует совместить расписания, построенные для всех интервалов I_j ($j = 1, 2, \ldots, p$).

Определим вычислительную сложность отдельных этапов и всего алгоритма. Построение потоковой сети G – $O(n^2)$; нахождение циркуляции минимальной стоимости с помощью алгоритма дефекта – $O(n^4 \sum_{i \in N} d_i)$; построение расписания в интервале I_j ($1 \le j \le 2n - 1$) – $O(n)$. Таким образом, вычислительная сложность алгоритма – $O(n^4 \sum_{i \in N} d_i)$.

3.1.4. Коррекция директивных интервалов

В настоящем разделе рассматривается задача, нахождения директивных интервалов $(b_i, f_i']$ (при фиксированной величине R), при которых (1) существует распределение ресурсов $(r_1^0, r_2^0, \ldots, r_n^0)$, удовлетворяющее ограничениям (1), (2); (2) для этого распределения ресурсов существует допустимое расписание; (3) величина $\max_{i \in N}(f_i' - f_i)$ принимает минимальное значение. Предположим, что при заданной величине ресурса R допустимого расписания не существует (т.е. $c(g) > R - \sum_{i \in N} d_i / a_i$). Поскольку общее количество ресурса фиксировано, то добиться выполнения неравенства (9) за счет увеличения величины R нельзя.

В дальнейшем будем предполагать, что директивные интервалы имеют вид $(0, f_i]$, $i \in N$. Без ограничения общности можно считать, что $f_i < f_{i+1}$ при всех $i \in N$, т.е. работы упорядочены по возрастанию правых границ их директивных интервалов. Построим для этого случая потоковую сеть и будем обозначать ее G^* (рис. 4). Заметим, что $I_1 = (0, f_1], I_2 = (f_1, f_2], \ldots, I_n = (f_{n-1}, f_n]$. Таким образом, особенностью данной сети является то, что из узла I_j исходят дуги, ведущие в узлы $\{w_j, w_{j+1}, \ldots, w_n\}$. Пусть g – циркуляция минимальной стоимости в

сети G^*, а $c(g)$ – ее стоимость. Обозначим потоки по дугам $\langle s, I_j \rangle$, $\langle I_j, w_i \rangle$ и $\langle w_i, t \rangle$ через $g_{\langle s, I_j \rangle}$, $g_{\langle I_j, w_i \rangle}$ и $g_{\langle w_i, t \rangle}$ соответственно.

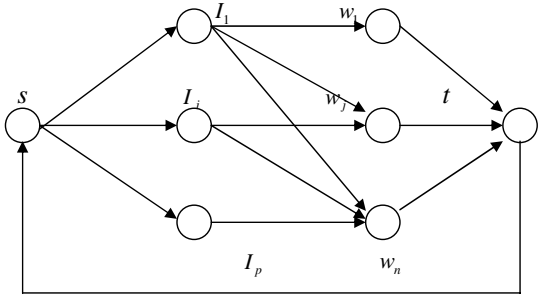


Рис. 4. *Потоковая сеть* G^*.

Увеличим правые границы директивных интервалов всех работ на величину $\varepsilon > 0$: $f_i' = f_i + \varepsilon$. Из постановки задачи следует, что правые границы всех директивных интервалов можно увеличивать на одну и ту же величину. Получим, что $\Delta_1^* = \Delta_1 + \varepsilon$, $\Delta_j^* = \Delta_j$, $j \in N \setminus \{1\}$. Это означает, что после такого увеличения всех директивных интервалов изменились пропускные способности дуг $\langle s, I_1 \rangle$ и $\langle I_1, w_i \rangle$, $i \in N$. Увеличение верхней границы пропускной способности дуг $\langle I_1, w_i \rangle$, $i \in N$ означает возможность увеличения потока по дугам $\langle w_i, t \rangle$, а поскольку стоимость потока по дугам $\langle w_i, t \rangle$ – величина отрицательная, то увеличение потока по ним снижает его стоимость.

Пусть $\delta = c(g) - \left[R - \sum_{i \in N} \frac{d_i}{a_i} \right]$. В дальнейшем будем рассматривать сеть G^*. Обозначим через $g^*_{\langle I_1, w_i \rangle}$ модифицированный поток по дугам $\langle I_1, w_i \rangle$. По дугам вида $\langle I_j, w_i \rangle$ $(j \geq 2)$ поток изменяться не будет. Имеем

$$\begin{aligned} g^*_{\langle I_1, w_i \rangle} &= g_{\langle I_1, w_i \rangle} + x_i, \\ g^*_{\langle I_j, w_i \rangle} &= g_{\langle I_j, w_i \rangle}, j \in N \setminus \{1\}. \end{aligned} \qquad (12)$$

Из неравенств $g_{\langle I_1, w_i\rangle} \le \Delta_1$ и $g^*_{\langle I_1, w_i\rangle} \le \Delta^*_1 = \Delta_1 + \varepsilon$ получаем первое ограничение на величины x_i:

$$x_i \le \Delta_1 + \varepsilon - g_{\langle I_1, w_i\rangle}. \tag{13}$$

Из условия сохранения потока в каждом узле сети следует, что величина потока, входящего в вершину I_1, изменилась на величину $\sum_{i\in N} x_i$, откуда следует второе ограничение:

$$\sum_{i\in N} x_i \le m\varepsilon. \tag{14}$$

Поскольку ненулевую стоимость потока имеют только дуги $\langle w_i, t\rangle$, то общая стоимость потока изменится на величину $\sum_{i\in N} x_i \cdot \left(-\frac{1}{a_i}\right)$, откуда следует третье ограничение на x_i:

$$\sum_{i\in N} x_i \cdot \left(-\frac{1}{a_i}\right) + \delta \le 0. \tag{15}$$

Минимизируя ε при указанных ограничениях (13) – (15), получаем следующую задачу линейного программирования:

$$\begin{aligned}
&\varepsilon \to \min,\\
&x_i \le \Delta_1 + \varepsilon - g_{\langle I_1, w_i\rangle}, \quad i \in N,\\
&\sum_{i\in N} x_i \le m\varepsilon,\\
&\sum_{i\in N} x_i \cdot \left(-\frac{1}{a_i}\right) + \delta \le 0,\\
&\varepsilon > 0, x_i \ge 0, i \in N.
\end{aligned}$$

Для произвольного подмножества $\overline{N} \subseteq N$ определим множество номеров l интервалов I_l $I(\overline{N}) = \{l : 1 \le l \le p$, существует $i \in \overline{N}$, $I_l \in (b_i, f_i]\}$ и величину $\overline{n}(l) = \left|\left\{i \in \overline{N} : I_l \in (b_i, f_i]\right\}\right|$. Как следует из [1], допустимое расписание существует тогда и только тогда, когда для любого подмножества $\overline{N} \subseteq N$ выполняется неравенство

$$\sum_{i\in \overline{N}} t_i \le \sum_{l \in I(\overline{N})} \Delta_l \min\left(m, \overline{n}(l)\right). \tag{16}$$

При этом достаточно рассмотреть только такие подмножества $\overline{N}$, для которых $I(\overline{N}) = \{l_1, l_1 + 1, ..., l_2\}$ при некоторых l_1, l_2. Поскольку директивные интервалы имеют вид $(0, f_i]$, $i \in N$ и $f_i < f_{i+1}$ при всех $i \in N$, достаточно рассматривать только подмножества $\overline{N}$ следующего вида: $\{1\}$, $\{1,2\}$, $\{1,2,3\}$, ..., $\{1,2,...,n\}$. Для подмножества $\{1\}$ условие (16) имеет вид $t_1 \le \Delta_1$; для подмножества $\{1,2\}$ – $t_1 + t_2 \le \Delta_1 \cdot 2 + \Delta_2$. В общем случае для подмножества $\{1,2,...,k\}$ имеем $\sum_{i=1}^{k} t_i \le \sum_{i=1}^{k} \Delta_i \cdot \min(m, k - i + 1)$. Увеличим правые границы директивных интервалов всех работ на ε. Получим, что $\Delta_1^* = \Delta_1 + \varepsilon$ и $\Delta_j^* = \Delta_j$, $j \in N \setminus \{1\}$. Таким образом, для некоторого фиксированного распределения ресурсов, которое можно найти, например, с помощью алгоритма, описанного в разд. 3.1.2, получаем следующую задачу линейного программирования:

$$\varepsilon \to \min,$$

$$\sum_{i \in N} r_i \le R,$$

$$\sum_{i=1}^{k} t_i \le \sum_{i=1}^{k} \Delta_i^* \cdot \min(m, k - i + 1), k = \overline{1,n}.$$

Выше задача поиска модифицированного потока была сведена к задаче линейного программирования. В настоящем разделе мы рассмотрим вопрос о приближенном решении поставленной задачи. Суть алгоритма заключается в последовательном насыщении дуг в соответствии с определенным приоритетом. Действительно, при достаточно большом значении ε для выполнения условия (3) можно ограничиться насыщением потока лишь по одной дуге. При уменьшении ε число таких дуг может возрасти. Фрагмент сети G^*, который будет рассматриваться в дальнейшем, изображен на рис. 5.

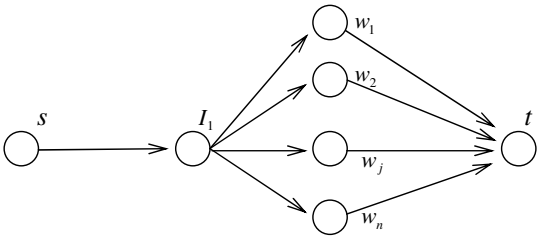


Рис. 5. Фрагмент сети G^*.

Пусть по дугам $\langle s, I_1\rangle, \langle I_1, w_i\rangle, i \in N$ уже имеется некоторый поток (например, поток, который был получен для проверки условия (3)). Обозначим $\alpha_0 = m\Delta_1 - g_{\langle s,I_1\rangle}$ и $\alpha_i = \Delta_1 - g_{\langle I_1,w_i\rangle}$, $i \in N$. Величины α_0 и α_i характеризуют максимально допустимое увеличение потока по дугам $\langle s, I_1\rangle, \langle I_1, w_i\rangle, i \in N$. Отсюда следует, что величины $\left[-\frac{\alpha_i}{a_i}\right]$ характеризуют максимально возможное уменьшение стоимости потока по дугам $\langle w_i, t\rangle$. Без ограничения общности можно считать, что выполнено следующее соотношение:

$$\frac{\alpha_1}{a_1} \geq \frac{\alpha_2}{a_2} \geq \ldots \geq \frac{\alpha_n}{a_n}. \tag{17}$$

В случае, когда условие (9) не выполнено, предлагаемый алгоритм легко может быть соответствующим образом модифицирован.

Рассмотрим случай, когда для выполнения условия (9) достаточно полностью насытить одну дугу. Пусть $\varepsilon_1 = \delta \cdot a_1$. Назначим по дуге $\langle I_1, w_1\rangle$ дополнительно $x_1 = \delta a_1 + \alpha_1$ единиц потока, а по дугам $\langle I_1, w_i\rangle, i \neq 1$ дополнительный поток не назначается, т.е. $x_i = 0$. Суммарная стоимость всего потока уменьшится на величину $\sum_{i\in N}\frac{x_i}{a_i} = \delta + \frac{\alpha_1}{a_1} \geq \delta$. Это значит, что $c(g) - c(g^*) \geq \delta$. Поскольку $\delta = c(g) - \left[R - \sum_{i\in N}\frac{d_i}{a_i}\right]$, то $c(g^*) \leq R - \sum_{i\in N}\frac{d_i}{a_i}$, а значит допустимое расписание существует.

На первом шаге величина $\varepsilon_1 = \delta \cdot a_1$ может оказаться достаточно большой и при этом полного насыщения достигает только дуга $\langle I_1, w_1 \rangle$, в то время как для дуги $\langle s, I_1 \rangle$ $\alpha_0 = (m-1)\delta \cdot a_1 - \alpha_1$, т.е. дуга $\langle I_1, w_1 \rangle$ насыщается не полностью. На этом шаге будем предполагать, что насыщаются k наиболее приоритетных дуг, т.е. дуги $\langle I_1, w_1 \rangle, ..., \langle I_1, w_k \rangle$. Насыщая большее количество дуг, величина ε уменьшается. На *k-м шаге* $\varepsilon_k = (\delta - \sum_{i=1}^{k} \frac{\alpha_i}{a_i}) / \sum_{i=1}^{k} \frac{1}{a_i}$. Аналогично тому, как показано на шаге 1, можно показать, что $c(g^*) \le R - \sum_{i \in N} \frac{d_i}{a_i}$. Условие $k < m$ необходимо для того, чтобы, насыщая k дуг, поток по дуге $\langle s, I_1 \rangle$ не превысил ее пропускную способность. Суммарный поток по дугам $\langle I_1, w_1 \rangle, ..., \langle I_1, w_k \rangle$ может увеличиться не более чем на величину $k\varepsilon + \sum_{i=1}^{k} \alpha_i$, а по дуге $\langle s, I_1 \rangle$ – не более чем на $m\varepsilon + \alpha_0$, где $\alpha_0 \ge \sum_{i=1}^{n} \alpha_i \ge \sum_{i=1}^{k} \alpha_i$.

Для ε_m имеем

$$\varepsilon_m = \frac{\delta - \sum_{i=1}^{m} \frac{\alpha_i}{a_i}}{\sum_{i=1}^{m} \frac{1}{a_i}}, \quad g_{\langle I_1, w_m \rangle} = \varepsilon_m - \max\left(-\alpha_m, \varepsilon_m - \alpha_0 + \sum_{k=1}^{m} \alpha_m \right).$$

Перейдем теперь к описанию алгоритма.

Шаг 1. Положить $\varepsilon = \delta \cdot a_1$, $x_1 = \delta a_1 + \alpha_1$. (Как показано выше, в этом случае $c(g^*) \le R - \sum_{i \in N} \frac{d_i}{a_i}$.)

Шаг 2. Положить $\varepsilon = \varepsilon / 2$.

Шаг 3. Если $\varepsilon \le \varepsilon_m$, перейти к *шагу 4.* Если $\varepsilon > \varepsilon_m$, перейти к *шагу 2.*

Шаг 4. Насытить дуги в порядке приоритета, пока выполнено условие $m\varepsilon + \alpha_0 \geq \sum_{i=1}^{s} x_i, s \leq m$. Если последнее условие не выполнено и $c(g^*) \leq R - \sum_{i \in N} \frac{d_i}{a_i}$, перейти к *шагу 6,* в противном случае – к *шагу 5.*

Шаг 5. Положить $\varepsilon = 1{,}5\varepsilon$. Перейти к *шагу 3.*

Шаг 6. Завершение алгоритма.

Данный алгоритм позволяет в случае, когда *m* не намного меньше *n*, с небольшими вычислительными затратами получать хорошие оценки для ε. В табл. 2 показано, на сколько процентов значения ε, полученные приближенным алгоритмом, отличаются от точных значений, полученных методом, описанным в разд. 3.1.2.

Таблица 2. Сравнение точного и приближенного методов

Параметры		Приближенный
m/n	δ	Отклонение (%)
0,01	10	300-1000
0,1	10	100-400
0,5	10	10-120
0,9	10	1-20
0,1	1	80-200
0,9	1	10-35
0,1	0,1	40-90
0,9	0,1	1-8

3.2. Построение периодических многопроцессорных расписаний с прерываниями

В настоящем разделе рассматривается задача построения допустимого расписания с прерываниями для случая, когда имеется несколько различных по производительности процессоров, работы допускают прерывания, а требования на выполнение работ поступают

циклически с заданными периодами. Случай циклического выполнения работ на одном процессоре описан в [10, 14].

3.2.1. Постановка задачи

Рассматривается вычислительная система, состоящая из M параллельных процессоров L типов. Число процессоров j-го типа равно m_j ($j = 1, 2, ..., L$; $\sum_{j=1}^{L} m_j = M$). Производительность процессоров j-го типа равна s_j ($j = 1, 2, ..., L$), $s_1 > s_2 > ... > s_L$. На вход системы поступает детерминированный поток требований на выполнение работ из множества $N = \{1, 2, ... , n\}$. Требования на обслуживание работы $i \in N$ поступают циклически с периодом $p_i > 0$, начиная с момента времени $p_i^o \geq 0$; k-е требование поступает в момент времени $p_i^0 + (k - 1)\, p_i$, $k = 1, 2, ...$ и для него установлен директивный срок $p_i^0 + (k - 1)p_i + f_i$ ($f_i \leq p_i$). Каждая работа может обслуживаться любым процессором. В фиксированный момент времени каждым процессором может выполняться не более одной работы, а каждая работа обслуживается не более чем одним процессором. Допускаются прерывания работ и переключения их с одного процессора на другой. Задан суммарный объем Q_i работы процессоров, необходимый для однократного исполнения каждой работы $i \in N$, т.е. если τ_{ij}^k – суммарное время обслуживания процессором j-го типа k-го требования работы i, то $Q_i = \sum_{j=1}^{L} s_j \tau_{ij}^k$. Требуется определить, существует ли допустимое расписание выполнения работ (т.е. расписание, при котором каждое требование работы i обслуживается в своем директивном интервале и суммарный объем работы процессоров по его обслуживанию составляет Q_i), и если оно существует, то найти его.

На практике подобные задачи возникают в тех случаях, когда с исследуемого объекта в АСУ периодически поступают измеряемые данные, являющиеся входными параметрами вычислительных модулей $i \in N$. Каждый модуль должен обрабатывать поступающую ин-

формацию с заданным периодом p_i. Кроме того, для отображения обработанной информации требуется время, равное $p_i - f_i$. При решении практических задач величины p_i обычно бывают рациональными. Поэтому, выбрав соответствующую единицу измерения, будем считать их целыми.

3.2.2. Алгоритм решения задачи

Расписание будем искать в виде вектор-функции $R(t) = (R_1(t), R_2(t), \ldots, R_n(t))$, где $R_i(t) = j$, если в момент времени t работа i обслуживается одним из процессоров j-го типа, и $R_i(t) = 0$, если работа i в момент времени t не выполняется. Введем следующие обозначения: P – наименьшее общее кратное величин $p_1, p_2, \ldots, p_n$; $p_0 = \max\{ p_i^0 : i \in N\}$; $B_m = (p_0 + (m\text{-}1)P, p_0 + mP]$ $(m = 1, 2,\ldots)$; U_{ih}^m $(m = 1, 2,\ldots; i \in N; h = 0, 1, \ldots, r_i+1)$ – интервалы, образованные пересечением B_m с интервалами P_{ki} $(k = 1, 2, \ldots)$ (если $p_0 = p_i^0$ или $p_0 - p_i^0$ кратно p_i, то полагаем $U_{i0}^m = \varnothing$); Ω – множество расписаний $R^o(t)$ на B_1, для которых $a_{ih} = Q_i$ при всех $i \in N$, $h = 1, 2, \ldots, r_i$ и $a_{i0} + a_{i(r\,i+1)} = Q_i$ при всех $i \in N$, где $a_{ih} = \sum_{j=1}^{L} s_j \tau_{ij}^h$, τ_{ij}^h – суммарная длина интервалов, на которых $R_i^o(t) = j$ и которые лежат в U_{ih}^l.

Если $\Omega \neq \varnothing$, то для произвольного $R^o(t) \in \Omega$ можно определить допустимое периодическое с периодом P расписание $R(t)$ следующим образом:

$$R_i(t) = \begin{cases} 0 \quad при \quad t \le p_i^0, \\ R_i^o(t + mP) \; при \; t > p_i^0, \end{cases} \tag{18}$$

где m – такое целое, что $t + mP \in B_1$, $i \in N$. Такое расписание будем называть P-периодическим. Опишем способ построения расписания $R^o(t) \in \Omega$ и сформулируем условие, при котором $\Omega \neq \varnothing$. Воспользуемся методом, аналогичным тому, который разработан в [11]. Пусть $d_0 < d_1 < \ldots < d_{p+1}$ – все различные элементы множества чисел

$$\{ \varphi_{ik} = p_i^0 + (k-1)p_i, \psi_{ik} = p_i^0 + (k-1)p_i + f_i, p_0 \le \varphi_{ik}, \psi_{ik} \le p_0 + P \},$$

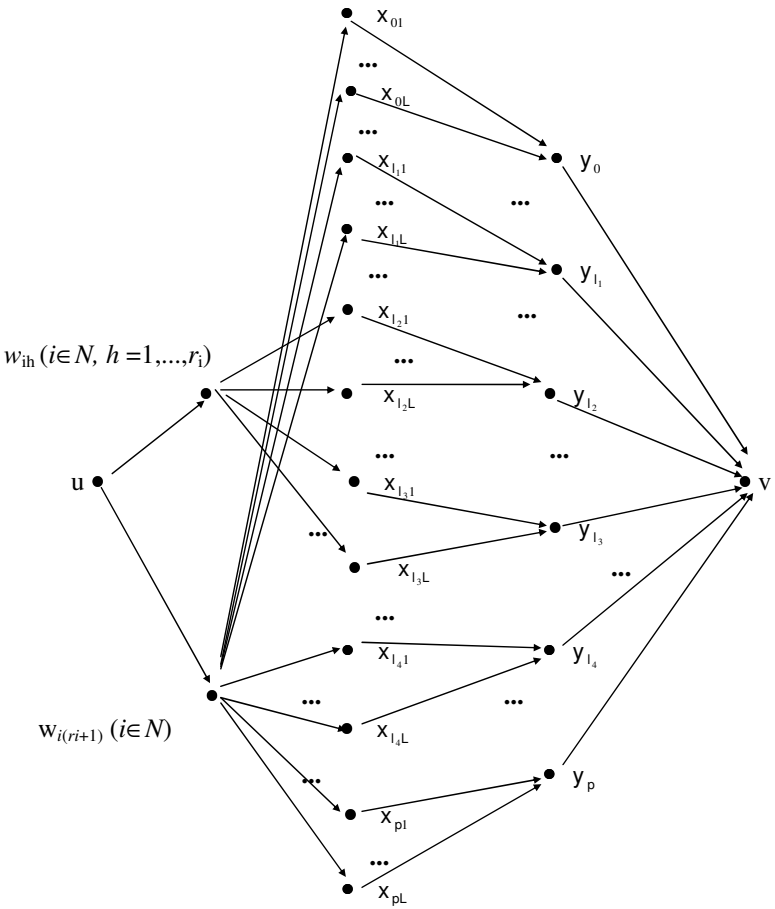
$i \in N$, $k = 1, 2, ...$, $D_l = (d_l, d_{l+1}]$, δ_l - длина интервала D_l ($l = 0, 1, ..., p$). Для каждой пары интервалов D_l и U^l_{ih} либо $D_l \subseteq U^1_{ih}$, либо $D_l \bigcap U^1_{ih} = \varnothing$. В дальнейшем среди интервалов D_l будем рассматривать только те, которые целиком лежат в некотором U^1_{ih} ($i \in N$, $0 \le h \le r_i + 1$). Без ограничения общности можно считать, что все D_l ($l = 0, 1, ..., p$) удовлетворяют этому требованию. Определим потоковую сеть G (рис. 6) с источником u, стоком v и внутренними узлами w_{ih} ($i \in N$, $h = 1, 2, ..., r_i + 1$), x_{lj} ($l = 0, 1, ... p$; $j = 1, 2, ..., L$) и y_l ($l = 0, 1, ..., p$) следующим образом: узел w_{ih} соответствует интервалу U^1_{ih}, узел $w_{i(ri+1)}$ – объединению интервалов U^1_{i0} и $U^1_{i(ri+1)}$, узел x_{lj} – интервалу D_l и процессору j-го типа, узел y_l – интервалу D_l; дуги (u, w_{ih}) существуют для всех $i \in N$, $h = 1, 2, ..., r_i + 1$ и имеют пропускную способность Q_i; дуги (w_{ih}, x_{lj}) существуют для всех $i \in N$, $h = 1, 2, ..., r_i$, $j = 1, 2, ... , L$ и тех l, при которых $D_l \subseteq U^1_{ih}$, и имеют пропускную способность $(s_j - s_{j+1})\delta_l$ (полагаем s_{L+1}=0); дуги $(w_{i(ri+1)}, x_{lj})$ существуют для всех $i \in N$, $j = 1, 2, ..., L$ и тех l, при которых $D_l \subseteq U^1_{i0} \bigcup U^1_{i(ri+1)}$, и имеют пропускную способность $(s_j - s_{j+1})\delta_l$; дуги (x_{lj}, y_l) существуют для всех $l = 0, 1, ..., p$, $j = 1, 2, ..., L$ и имеют пропускную способность $\sum_{r=1}^{j} m_r(s_j - s_{j+1})\delta_l$; дуги (y_l, v) существуют для всех $l = 0, 1, ..., p$ и имеют неограниченную пропускную способность.

Лемма 2. $\Omega \neq \varnothing$ в том и только том случае, когда максимальный поток в сети G насыщает все дуги (u, w_{ih}) ($i \in N$, $h = 1, 2, ..., r_i + 1$).

Доказательство этой леммы аналогично доказательству соответствующего утверждения в [11].

Интерпретируя суммарную величину потока из узла w_{ih} в узел y_l как суммарный объем работы процессоров по выполнению работы i в интервале D_l, с помощью алгоритма, предложенного в [12], можно построить расписание выполнения заданий в каждом интервале D_l.

Таким образом, для построения допустимого расписания $R(t)$ достаточно найти некоторое $R^o(t) \in \Omega$ и определить $R(t)$ согласно (18).



Узлы $y_0, ..., y_{l_1}$ *соответствуют интервалу* U^1_{i0}, *узлы* $y_{l_2}, ... y_{l_3}$ – *интервалу* U^1_{ih}, *узлы* $y_{l_4}, ..., y_p$ – *интервалу* $U^1_{i(ri+1)}$.

Рис. 6. Потоковая сеть, используемая для построения расписания.

В свою очередь, для определения $R^o(t) \in \Omega$ надо построить сеть G и вычислить максимальный поток. Если при этом все дуги (u, w_{ih}) $(i \in N, h = 1, 2, ..., r_i + 1)$ окажутся насыщенными, то полу-

ченный максимальный поток определит некоторое расписание $R^o(t) \in \Omega$. Если же максимальный поток не насыщает хотя бы одну дугу (u, w_{ih}), то согласно лемме 2 имеем $\Omega = \varnothing$. В этом случае P-периодического расписания не существует. Как следует из следующей леммы, основанной на результатах работы [4], никакого другого допустимого расписания в этом случае не существует.

Лемма 3. Допустимое расписание существует в том и только том случае, когда существует допустимое P-периодическое расписание.

Как следует из [12], число прерываний в каждом интервале D_l ($l = 0, 1, ..., p$) не превосходит $2(M - 1)$, а общее число прерываний в P-периодическом расписании, построенном указанным выше способом, в каждом интервале B_m не превосходит $2(M-1)(p+1)+ pn$ - $-P\sum_{i=1}^{n}(1/p_i)$.

Вычислительная сложность различных этапов предложенного алгоритма следующая: построение сети $G - O(L(P\sum_{i=1}^{n}(1/p_i)^2)$; нахождение максимального потока [116] $- O((LP\sum_{i=1}^{n}(1/p_i))^3)$; построение расписания по найденному максимальному потоку внутри каждого интервала D_l [12] $- O(n + M \log M)$, а внутри каждого интервала $B_m - O((n + M \log M) P\sum_{i=1}^{n}(1/p_i))$. Таким образом, вычислительная сложность предложенного алгоритма $O((LP\sum_{i=1}^{n}(1/p_i))^3)$.

3.3. Составление допустимых расписаний без прерываний при наличии неопределенных факторов

Рассматривается задача распределения возобновляемых ресурсов в системе реального времени при ограничении на время выполнения заданной совокупности непрерываемых работ. В некоторые неопределенные моменты времени имеющиеся ресурсы могут быть переданы более приоритетным работам. В результате этого выполнение исход-

ной совокупности работ прекращается и переносится на более позднее время; тем самым нарушается построенное ранее расписание. Выработана такая стратегия построения допустимых расписаний, при которой вероятность их нарушения минимальна.

3.3.1. Постановка задачи

Задана совокупность непрерываемых работ $W = \{w_1, w_2, \ldots, w_n\}$ и частичный порядок их выполнения, определяемый графом без циклов $G=(W,E)$, где W – вершины, E – дуги (если $(w_i, w_j) \in E$, то работа w_j может быть начата только после окончания работы w_i; для такого отношения частичного порядка будем также использовать обозначение $w_i \rightarrow w_j$). При выполнении работ используются ресурсы q типов; R_j - количество ресурсов типа j ($j = 1, 2, \ldots, q$). Известны следующие характеристики каждой работы $w_i \in W$: $t(w_i)$ - длительность выполнения, r_{ij} - количество ресурсов типа j, необходимое для ее выполнения ($r_{ij} \le R_j$). Задан директивный срок T, не позднее которого необходимо завершить все работы W ($T \ge T^*$, где T^* - минимальное время выполнения всех работ). В некоторые неопределенные моменты времени $y_1, y_2, \ldots, y_m$ могут поступать запросы на выполнение более приоритетных работ $V = \{v_1, v_2, \ldots, v_m\}$ (m задано, $0 \le y_1 \le y_2 \le \ldots \le y_m \le T$). Каждая работа v_j выполняется без прерываний, ее длительность составляет $t(v_j)$ и для нее выделяются все имеющиеся ресурсы. Если в интервале $[y_j, y_j + t(v_j)]$ выполняется некоторая работа $w_i \in W$, то ее выполнение прекращается и переносится на более позднее время. Тем самым нарушается построенное ранее расписание выполнения работ W. Процесс выполнения работ W и поступления запросов на выполнение работ V повторяется многократно. Задача заключается в выработке стратегии построения таких расписаний выполнения работ W, которые удовлетворяют следующим требованиям:

1. Не нарушаются ограничения на последовательность выполнения работ W, задаваемые графом G.

2. Не нарушаются ограничения на использование ресурсов.

3. Если запросы на выполнение работ V не поступают в интервале $[0, T)$, то время выполнения совокупности работ W не превосходит T.

4. Вероятность того, что расписание для W будет нарушено вследствие поступления запросов на выполнение работ V, минимальна.

Опишем формально условия 1 – 3. Расписание выполнения работ W будем задавать в виде совокупности $S_\alpha = (\alpha(w_1), \alpha(w_2), \ldots, \alpha(w_n))$, где $\alpha(w_i)$ – момент начала выполнения работы w_i, а также в виде совокупности $S_\theta = (W_1, W_2, \ldots, W_p, \theta_1, \theta_2, \ldots, \theta_p)$. Здесь работы $W_1 \subseteq W$ выполняются в течение интервала времени θ_1, затем работы $W_2 \subseteq W$ выполняются в течение интервала времени θ_2 и т.д. Кроме того, все работы из W_k, которые остались незавершенными после момента времени $\theta_1 + \theta_2 + \ldots + \theta_k$, входят в W_{k+1} $(k = 1, 2, \ldots, p - 1)$. Через $\theta(S)$ будем обозначать время выполнения совокупности работ W по расписанию S_θ, т.е. $\theta(S) = \sum_{k=1}^{p} \theta_k$. В этих обозначениях условия 1 – 3 могут быть записаны так:

1. $\alpha(w_i) + t(w_i) \le \alpha(w_j)$ для всех i, j таких, что $(w_i, w_j) \in E$;

2. $\sum_{i:\alpha(w_i) \le t \le \alpha(w_i)+t_i} r_{ij} \le R_j$ при всех $t \in [0, T]$ и $j = 1, 2, \ldots, q$;

3. $\theta(S) \le T$.

3.3.2. Обсуждение задачи

Алгоритм решения задачи состоит из двух этапов. На первом этапе строится множество X векторов $x = (x_1, x_2, \ldots, x_n)$, для каждого из которых существует такое расписание S_x, что не нарушаются условия 1 – 3 и, кроме того, $\alpha(w_i) = x_i$. На втором этапе рассматривается антагонистическая игра с платежной функцией

$$K(x,y)=\begin{cases}1, & \textit{если} \quad x_i + t(w_i) \le y_j \quad \textit{или} \quad x_i \ge y_j + t(v_j) \\ & \textit{при всех} \quad i=1,2,...,n;\ j=1,2,...,m; \\ 0 & \textit{в противном случае},\end{cases} \qquad (19)$$

определенной на множествах $X \times Y$, где $Y = \{y = (y_1, y_2, \ldots, y_m)$: $0 \le y_1 \le y_2 \le \ldots \le y_m \le T\}$ описывает все возможные моменты поступления запросов на выполнение работ V. Если $K(x, y) = 1$, то работы V могут быть выполнены так, что при этом расписание S_x для W не будет нарушено. Если же $K(x, y) = 0$, то работы W не могут быть выполнены по расписанию S_x. Пусть $f(x)$ – некоторая смешанная стратегия первого игрока в игре с платежной функцией $K(x, y)$. Тогда при фиксированном $y \in Y$ величина $\int_X K(x,y)df(x)$ есть вероятность того, что расписание для W не будет нарушено. Оптимальная смешанная стратегия $f^o(x)$ первого игрока максимизирует величину $\inf_{y \in Y} \int_X K(x,y)df(x)$ и определяет искомую стратегию построения расписаний S_x. Метод решения игры (19), основанный на аппроксимации ее конечными играми, описан в [117].

3.3.3. Построение множества X

Введем обозначения: $P(w_i)$ - множество всех работ $w_j \in W$, предшествующих (не обязательно непосредственно) работе w_i в графе G; $Q(w_i)$ – множество всех работ $w_k \in W$, следующих (не обязательно непосредственно) за работой w_i в графе G (т.е. для каждой работы $w_j \in P(w_i)$ существует ориентированный путь из w_j в w_i в графе G, а для каждой работы $w_k \in Q(w_i)$ существует ориентированный путь из w_i в w_k); $\tau(W')$ – минимальное время выполнения совокупности работ W' с учетом ограничений по ресурсам и частичному порядку их выполнения (эта величина может быть вычислена, например, по алгоритму, предложенному в [118]); $A_i^1 = \tau(P(w_i))$; $A_i^2 = T - t(w_i) - \tau(Q(w_i))$. Величины A_i^1 и A_i^2 являются соответственно наиболее ранним возможным и наиболее поздним допустимым срока-

ми начала выполнения работы w_i. Поскольку $\tau(P(w_i)) + t(w_i) + \tau(Q(w_i)) \le T^*$ и $T \le T^*$, то $A_i^1 \le A_i^2$.

Для вектора $x = (x_1, x_2, \ldots, x_n)$, $x_i \in [A_i^1, A_i^2]$, $i = 1, 2, \ldots, n$, определим множество работ $W_x' = W \cup \{w_i^1, w_i^2\}$, $i = 1,2,\ldots,n$, где w_i^1, w_i^2 – фиктивные работы (не требующие ресурсов), $t(w_i^1) = x_i$, $t(w_i^2) = T - t(w_i) - x_i$; на множестве W сохраняется отношение частичного порядка, задаваемое графом G, и, кроме того, $w_i^1 \to w_i \to w_i^2$.

Лемма 4. Соотношение $x \in X$ имеет место тогда и только тогда, когда $\tau(W_x') \le T$.

Доказательство. 1. Пусть $x \in X$. Тогда для W существует расписание S_x, при котором не нарушаются условия 1 – 3 и $\alpha(w_i) = x_i$ для всех $w_i \in W$. Построим расписание S'_x для W'_x, дополнив S_x работами w_i^1 и w_i^2 и определив $\alpha(w_i^1) = 0$, $\alpha(w_i^2) = x_i + t(w_i)$. Тогда время выполнения совокупности работ W'_x равно T, т.е. неравенство $\tau(W'_x) \le T$ выполнено.

2. Пусть теперь $\tau(W'_x) \le T$. Тогда для W'_x существует расписание S'_x, при котором $\theta(S'_x) \le T$. Следовательно, и для W существует расписание, при котором выполняются условия 1 - 3. Поскольку $w_i^1 \to w_i$ и $t(w_i^1) = x_i$, то $\alpha(w_i) \ge x_i$. С другой стороны, поскольку $w_i \to w_i^2$, $t(w_i^2) = T - t(w_i) - x_i$ и $\alpha(w_i) + t(w_i) + t(w_i^2) \le T$, то $\alpha(w_i) + t(w_i) + t(w_i^2) = \alpha(w_i) + t(w_i) + T - t(w_i) - x_i \le T$, т.е. $\alpha(w_i) \le x_i$. Следовательно, $\alpha(w_i) = x_i$ и $x \in X$. Лемма доказана.

Пусть $B_i^1 = T - t(w_i) - A_i^2 = T - t(w_i) - (T - t(w_i) - \tau(Q(w_i))) = \tau(Q(w_i))$; $B_i^2 = T - t(w_i) - A_i^1 = T - t(w_i) - \tau(P(w_i))$. Поскольку $A_i^1 \le A_i^2$, то $B_i^1 \le B_i^2$. Рассмотрим совокупность работ W'_x с определенным выше отношением частичного порядка на нем и предположим, что $t(w^1_i) \in [A_i^1, A_i^2]$, $t(w_i^2) \in [B_i^1, B_i^2]$. Определим параллелепипед $\Pi_{2n} = [A_1^1, A_1^2] \times \ldots \times [A_n^1, A_n^2] \times \times [B_1^1, B_1^2] \times \ldots \times [B_n^1, B_n^2]$, лежащий в $2n$-мерном евклидовом пространстве с координатными осями, соответствующими переменным $t(w_i^1)$, $t(w_i^2)$, $i = 1, 2, \ldots, n$. Как следует из [119], существует конечное число гиперплоскостей Γ'_j вида

$$\sum_{i=1}^{n} g_i^j t(w_i^1) + \sum_{i=1}^{n} h_i^j t(w_i^2) = 0 \qquad (20)$$

(величины g_i^j и h_i^j принимают значения 0, 1 и -1), разбивающих параллелепипед Π_{2n} на выпуклые многогранники M'_1, M'_2, …, M'_Z, причем для каждого M'_k и вектора t_{2n}=($t(w_1^1)$,$t(w_2^1)$, …, $t(w_n^1)$, $t(w_1^2)$, $t(w_2^2)$, …, $t(w_n^2)$) $\in M'_k$ величина $\tau(W'_X)$ представляется в виде линейного функционала

$$L'(t_{2n}) = \sum_{i=1}^{n} c_i^K t(w_i^1) + \sum_{i=1}^{n} d_i^K t(w_i^2) \qquad (21)$$

(величины c_i^k, d_i^k принимают значения 0 или 1). Алгоритм построения гиперплоскостей Γ'_j и функционалов $L'_k(t_{2n})$ описан в [119].

Определим параллелепипед $P_n = [A_1^1, A_1^2] \times \ldots \times [A_n^1, A_n^2]$, лежащий в n-мерном евклидовом пространстве с координатными осями, соответствующими переменным $t(w_i^1)$, $i = 1, 2, \ldots, n$. Поскольку $t(w_i^2) = T - t(w_i) - t(w_i^1)$, то из (20), (21) следует, что гиперплоскости Γ_j вида

$$\sum_{i=1}^{n} g_i^j t(w_i^1) + \sum_{i=1}^{n} h_i^j (T - t(w_i) - t(w_i^1)) = 0, \text{ или}$$

$$\sum_{i=1}^{n} (g_i^j - h_i^j) t(w_i^1) + \sum_{i=1}^{n} h_i^j (T - t(w_i)) = 0,$$

разбивают параллелепипед Π_n на выпуклые многогранники M_1, M_2, …, M_Z, причем для каждого M_k и вектора t_n=($t(w_1^1)$,$t(w_2^1)$, …, $t(w_n^1)$) $\in M_k$ величина $\tau(W'_X)$ представляется в виде линейного функционала

$$L_k(t_n) = \sum_{i=1}^{n} (c_i^k - d_i^k) t(w_i^1) + \sum_{i=1}^{n} d_i^k (T - t(w_i)).$$

Для каждого $k = 1, 2, \ldots, Z$ гиперплоскостью $L_k(t_n) = T$ разобьем M_k на два таких выпуклых многогранника M_k^1 и M_k^2 (один из которых может оказаться пустым множеством), что $L_k(t_n) \le T$ при $t_n \in M_k^1$ и $L_k(t_n) > T$ при $t_n \in M_k^2$. Тогда $X = \bigcup_{k=1}^{Z} M_k^1$.

3.3.4. Пример

Рассмотрим совокупность из четырех работ $W = \{w_1, w_2, w_3, w_4\}$ с частичным порядком выполнения $w_1 \to w_2$, $w_2 \to w_3$. Имеются ресурсы двух типов ($q = 2$) в количестве $R_1 = 1$, $R_2 = 1$. Работы w_1, w_3, w_4 используют одну единицу ресурса первого типа, а работа w_2 - одну единицу ресурса второго типа. Задан директивный срок $T = 6$. В интервале времени [0, 6] может поступить запрос на выполнение одной более приоритетной работы v_1 ($m=1$, $V = \{v_1\}$), для которой требуется одна единица ресурса второго типа. Если этот ресурс будет занят работой w_2, то ее выполнение будет прекращено, а ресурс будет передан работе v_1. Длительности выполнения работ следующие: $t(w_1) = t(w_2) = t(w_3) = t(v_1) = 1$, $t(w_4) = 2$. Определим стратегию построения расписаний для W, при которой выполняются требования 1 – 4. Множество X в данном случае одномерное и состоит из всех моментов времени $x \in [0, 6]$, для каждого из которых существует расписание S_x, удовлетворяющее требованиям 1 – 3 и такое, что $\alpha(w_2) = x$. В рассматриваемом примере $X = [1, 2] \bigcup [3, 4]$, $Y = [0, 6]$. Рассмотрим антагонистическую игру с платежной функцией

$$K(x, y) = \begin{cases} 1, & \textit{если} \quad x+1 \le y \quad \textit{или} \quad x \ge y+1, \\ 0 & \textit{в} \quad \textit{противном} \quad \textit{случае}, \end{cases}$$

определенной на множестве $X \times Y$. Пусть $\nu_a(x)$ – вероятностная мера на X, сосредоточенная в одной точке $a \in X$, в которой имеет скачок, равный 1; $\mu_{[b,c]}(y)$ – вероятностная мера, равномерно распределенная на интервале $[b,c] \subseteq Y$. Тогда вероятностные меры

$$f_0(x) = 0{,}5\nu_1(x) + 0{,}5\nu_4(x) \text{ и } g_0(x) = 0{,}5\mu_{[1,2]}(y) + 0{,}5\mu_{[3,4]}(y)$$

являются оптимальными смешанными стратегиями первого и второго игроков соответственно, а значение игры равно 0,5.

Действительно, пусть

$$F(y) = \int_X K(x, y)df_0(x), \; G(x) = \int_Y K(x, y)dg_0(x).$$

Тогда $F(y) = 1$ при $y = 0$, $y \in [2, 3]$ и $y \in [5, 6]$; $F(y) = 0{,}5$ при $y \in (0, 2)$ и $y \in (3, 5)$; $G(x) = 0{,}5$ при всех $x \in [1, 2]$ и $x \in [3, 4]$. Таким образом, с вероятностью 0,5 следует использовать каждое из расписаний $S_{\alpha 1} = (0, 1, 2, 3)$ и $S_{\alpha 2} = (2, 4, 5, 0)$ для W. (Здесь i-я компонента вектора $S_{\alpha 1}$ или $S_{\alpha 2}$ равна моменту начала выполнения работы w_i.)

Глава 4.
Алгоритмы организации контроля в системах реального времени

Рассматривается задача оптимизации структуры подсистемы контроля в вычислительных системах реального времени. Разработаны алгоритмы, позволяющие определить такое расположение модулей контроля и модулей-буферов, при котором математическое ожидание суммарного времени, затраченного на выполнение всех модулей (включая повторное их выполнение при возникновении ошибок), минимальное. Доказательства приводимых в этой главе утверждений содержатся в [135, 136].

4.1. Введение

При функционировании вычислительных систем реального времени большое значение имеет подсистема организации рестартов, позволяющая в случае возникновения сбоев определить те программные модули, которые подлежат повторному запуску. Контроль поступающих в систему данных, а также данных, которыми обмениваются программные модули, осуществляется специальными модулями контроля, каждый из которых определяет, удовлетворяет ли множество значений контролируемых им параметров определенным условиям. Кроме того, в вычислительную систему реального времени вводятся дополнительные модули-буфера, сохраняющие промежуточную информацию для того, чтобы при рестарте системы воспользоваться данными из этих буферов, а не проводить все вычисления заново. Таким образом, наличие модулей контроля и модулей-буферов может существенно улучшить надежность системы и уменьшить временные задержки, вызванные несистематическими сбоями и ошибками. С другой стороны, избыточное использование этих модулей может привести к существенному уменьшению быстродействия и увеличению стоимости системы. Поэтому возникает задача оптимальной частоты расположения модулей контроля и модулей-буферов.

Задача оптимального расположения модулей контроля и модулей-буферов при различных условиях была широко исследована в работах [120 - 132]. В работах [120, 121] предполагалось, что модули-буфера надо расставлять равномерно, и получена оптимальная частота их расположения. В работе [120] вероятность возникновения сбоя или ошибки предполагалась малой, а в работе [121] величина оптимального интервала между последовательными модулями-буферами была найдена приближенно. Точная формула для длины оптимального интервала была получена в работах [122, 123]. В работе [124] рассматривается неравномерное расположение модулей-буферов для случая, когда вероятность возникновения ошибки не является постоянной. В работах [125, 126] получен алгоритм, позволяющий определять расположение модулей-буферов в предположении, что вероятность ошибки в одном и том же месте системы может изменяться после перезапуска. В работе [127] предлагается эвристическая расстановка модулей-буферов, основанная на увеличении интервала между последовательными модулями-буферами, если некоторое время не было ошибки. В работе [128] получена расстановка модулей-буферов в предположении, что вероятность возникновения ошибки не является постоянной и что ошибка может возникать во время записи данных в буфер. В работе [129] был применен метод стохастического динамического программирования для вычисления оптимального расположения модулей-буферов между модулями заданной длины. В [130] рассмотрена многопроцессорная система и получено расположение модулей-буферов, при котором вероятность выполнить работу до выхода из строя всех процессоров максимальна. В [131] получен алгоритм динамической расстановки модулей-буферов во время работы системы в зависимости от изменения обстоятельств. В работе [132] строится расстановка модулей-буферов при неполной информации о вероятности возможной ошибки.

Во всех этих работах, однако, сделано два упрощающих предположения, которые не всегда имеют место на практике. Во-первых, предполагается, что при возникновении ошибки можно сразу же ее обнаружить и выполнить перезапуск системы. На практике, однако, часто возникают также ошибки при вычислениях, которые невозмож-

но обнаружить сразу. В этом случае контроль системы осуществляется специальными модулями контроля, каждый из которых определяет, принадлежит ли множество значений контролируемых им параметров заданным пределам. При этом перезапуск системы происходит не сразу при возникновении ошибки, а только после ее обнаружения некоторым модулем контроля. Подсистема контроля данных подробно рассмотрена в [133].

Во-вторых, в работах [120 - 133] предполагается, что задания выполняются последовательно в строго оговоренном заранее порядке. На практике, однако, некоторые задания являются независимыми по данным, и при выполнении можно менять их местами. В общем случае все задания могут быть связаны произвольным графом зависимостей по данным. Такая модель организации рестартов в системах реального времени была рассмотрена Сушковым Б.Г. и Белым Д.В. в [134]. В этой работе вычислительная система реального времени рассматривается как граф, вершинами которого являются рабочие модули (которые и выполняют задания), модули контроля и модули-буфера. Кроме того, в [134] строится зона рестарта – минимальная область программы, подлежащая перезапуску после обнаружения ошибки некоторым модулем контроля. Именно эта модель взята за основу в настоящей работе. В [134], однако, предполагается, что модули контроля и модули-буфера уже расставлены некоторым образом.

В настоящей главе рассматривается задача оптимальной расстановки модулей контроля и модулей-буферов в заданном графе, вершинами которого являются рабочие модули.

4.2. Постановка задачи

Дан ориентированный граф $G = <V, E>$, вершины V которого – рабочие модули $M_1, M_2, \ldots, M_n$, ребра E – зависимости по данным между ними (ребро (i, j) означает, что модуль i передает данные модулю j). Рабочие модули – это главные модули системы, выполняющие вычисления. Время выполнения каждого рабочего модуля предполагается одинаковым и равным единице. При выполнении каждого рабочего модуля с вероятностью $a > 0$ может возникнуть ошибка, и тогда его

необходимо выполнить повторно. Помимо рабочих модулей в системе имеются модули контроля $C_i,\ i=1,2,...,k$, которые проверяют рабочие модули на наличие ошибок, и модули-буфера $B_i,\ i=1,2,...,m$, которые сохраняют полученные данные и передают их последующим модулям. Предполагается, что время работы каждого модуля контроля равно нулю, и он указывает на ошибку в том и только том случае, если хотя бы в одном из предшествующих ему по графу модулей произошла ошибка. Время работы каждого модуля-буфера также равно нулю. Для повторного выполнения рабочего модуля M_i нужно повторно выполнить все рабочие модули, которые непосредственно ему предшествуют, чтобы получить данные для M_i. Для каждого из его предшественников, в свою очередь, необходимо повторно выполнить рабочие модули, предшествующие ему, и т. д., пока не дойдем до модулей-буферов.

Рабочие модули должны выполниться без ошибки. Каждый модуль контроля и модуль-буфер может располагаться в начале или в конце некоторого рабочего модуля. Такое соответствие между модулями будем называть расстановкой. Требуется расположить модули контроля и модули-буфера так, чтобы математическое ожидание суммарного времени, затраченного на выполнение всех модулей (включая повторное их выполнение при возникновении ошибок), было минимальным. В дальнейшем такую расстановку модулей контроля и модулей–буферов будем называть оптимальной. Минимум ищется по всем допустимым расстановкам. Расстановка называется допустимой, если ошибка в любом рабочем модуле может быть обнаружена некоторым модулем контроля и все данные могут быть восстановлены. Иными словами, для каждого рабочего модуля существует ориентированный путь, ведущий из этого модуля в некоторый модуль контроля, и ориентированный путь, ведущий из некоторого модуля-буфера в этот рабочий модуль (в том числе учитываются и пути, состоящие из одной вершины, для случаев, когда модуль контроля помещен в конец рабочего модуля или модуль-буфер помещен в начало рабочего модуля). Предполагается, что имеется только один процессор, т. е. параллельные вычисления невозможны.

4.3. Расположение модулей контроля для случая одной цепочки рабочих модулей

В этом разделе будем предполагать, что выполнены следующие условия.

1. Граф G – это цепочка из n вершин, т.е. рабочие модули M_1, M_2, …, M_n соединены ребрами вида (M_i, M_{i+1}), $i = 1, 2, \ldots, n-1$.

2. Число рабочих модулей достаточно велико, т.е. $n >> k$. Это позволит считать, что с определенной степенью точности можно располагать модули контроля и модули-буфера в любой точке временного отрезка $[0, T]$, где T – это суммарное время выполнения всех модулей при условии, что ошибки не произойдет. При этих предположениях время выполнения рабочего модуля равно единице, модуля контроля или модуля-буфера – нулю, и, следовательно, $T = n$.

3. $a << 1/n$, т. е. ошибка возможна, но маловероятна. В этом случае можно считать, что при работе всей системы ошибка может произойти не более одного раза. Случаем, когда после перезапуска системы ошибка произойдет снова, мы пренебрегаем как маловероятным. Это позволит в дальнейшем существенно упростить вычисления и получить точные формулы для места расположения модулей контроля и модулей-буферов.

4.3.1. Предварительные результаты

В этом разделе мы сформулируем некоторые предварительные результаты, которые понадобятся в дальнейшем при построении оптимальной расстановки модулей контроля и модулей-буферов. Из условия допустимости расстановки следует, что необходимо расположить модуль-буфер перед первым рабочим модулем и модуль контроля после последнего. Остальные модули контроля и модули-буфера надо расставить оптимальным образом.

Предположим сначала, что модули контроля $C_1, C_2, \ldots, C_k$ и модули-буфера $B_1, B_2, \ldots, B_m$ уже расставлены. Здесь и далее, когда мы будем говорить о расстановке модулей контроля и модулей-буферов, мы будем нумеровать их в соответствии с порядком выполнения: для любых натуральных i, j, таких что $1 \le i < j \le k$, модуль контроля C_i

расположен перед модулем контроля C_j, а для любых i, j, таких что $1 \le i < j \le m$, модуль-буфер B_i расположен перед модулем-буфером B_j. Опишем множество рабочих модулей, выполнение которых следует повторить при возникновении ошибки. Для модуля контроля C_i через $B(C_i)$ обозначим ближайший предшествующий ему модуль-буфер.

Утверждение 1. Если некоторым модулем контроля C_i ($i > 1$) обнаружена ошибка, то следует повторить выполнение всех модулей от $B(C_{i-1})$ до C_i. При $i=1$ следует повторить выполнение всех модулей, расположенных до C_1.

Перейдем теперь к описанию свойств оптимальной расстановки. Будем обозначать некоторую допустимую расстановку k модулей контроля и m модулей-буферов в цепочке из n рабочих модулей через $A(k,m,n)$, а оптимальную расстановку – через $A_{опт}(k,m,n)$. Пусть $MA(k,m,n)$ (и соответственно $M_{опт}(k,m,n)$) – математическое ожидание суммарной длительности выполнения всех модулей (включая их повторное выполнение при возникновении ошибок). Следующая теорема показывает, что не имеет смысла иметь модулей-буферов больше, чем модулей контроля.

Теорема 1. При $k < m$ имеем $M_{опт}(k,m,n) = M_{опт}(k,k,n)$.

Эта теорема позволяет нам в дальнейшем без ограничения общности рассматривать только случай $k \ge m$. Будем говорить, что расположение некоторого вспомогательного модуля (т.е. модуля-буфера или модуля контроля) совпадает с расположением другого вспомогательного модуля, если между этими модулями не располагается ни один рабочий модуль. Следующая теорема позволяет существенно сузить класс расстановок, среди которых может оказаться оптимальная.

Теорема 2. При $k \ge m$ в оптимальной расстановке расположение каждого модуля-буфера (кроме первого) совпадает с расположением некоторого модуля контроля.

Эта теорема позволяет в дальнейшем искать оптимальную расстановку не среди всех расстановок, а только среди тех, в которых расположение каждого модуля-буфера (кроме первого) совпадает с

расположением некоторого модуля контроля. Следующая теорема показывает, что оптимальная расстановка является также и локально оптимальной (на каждом интервале между модулями-буферами).

Теорема 3. Пусть модули контроля $C_1, C_2,\ldots, C_k$ и модули-буфера $B_1, B_2,\ldots, B_m$ расставлены оптимальным образом в указанном порядке и пусть между некоторыми модулями-буферами B_i и B_{i+p} расположены r модулей контроля и s рабочих модулей. Тогда эти p модулей-буферов и r модулей контроля составляют оптимальную расстановку $A_{опт}(r, p, s)$ на рассматриваемом участке из s рабочих модулей.

Перейдем теперь непосредственно к построению оптимальных расстановок модулей контроля и модулей-буферов для различных случаев.

4.3.2. Случай $k = m$

Согласно теореме 1 можно без ограничения общности считать, что число модулей-буферов не превосходит числа модулей-контроля, т.е. $k \geq m$. Мы начнем с построения оптимальной расстановки модулей контроля и модулей-буферов для случая, когда $k = m$. Из условия допустимости расстановки следует, что первый модуль-буфер расположен в начале цепочки, а последний модуль контроля – в конце. Из теоремы 2 следует, что в случае $k = m$ в оптимальной расстановке расположение остальных $k-1$ модулей-буферов совпадает с расположением модулей контроля. Пусть $A(k,k,n)$ – произвольная расстановка, удовлетворяющая этому условию. Пусть при этой расстановке k модулей контроля разбивают рабочие модули на группы с числом модулей $x_1, x_2,..., x_k$ $(\sum_{i=1}^{k} x_i = n)$ соответственно. Вычислим для такой расстановки математическое ожидание времени выполнения $MA(k,k,n)$. Согласно предположению 3 разд.4.3 можно считать, что с точностью до малых более высокого порядка модуль контроля C_i обнаружит ошибку с вероятностью $a \cdot x_i$. Поскольку рядом с каждым модулем контроля находится модуль-буфер, то следует повторно выполнить x_i рабочих модулей. С вероятностью $1 - a \cdot n$ все рабочие мо-

дули отработают без ошибки. Тогда для искомого математического ожидания получаем:

$$MA(k,k,n) = (1 - a \cdot n)n + (a \cdot x_1)(n + x_1) + ...$$
$$+ a \cdot x_k (n + x_k) = n + \alpha \sum_{i=1}^{k} x_i^2. \quad (1)$$

Поскольку расстановка полностью задается числами $x_1, x_2, ..., x_k$, то достаточно минимизировать выражение (1) при условии $\sum_{i=1}^{k} x_i = n$. Несложно видеть, что минимум достигается при $x_i = n/k, i = 1,...,k$. Следовательно, в этом случае модули контроля и модули-буфера следует расположить равномерно. Заметим, что, вообще говоря, мы не можем осуществить такую расстановку, так как значение n / k не всегда является целым. Именно для этого было введено условие $n >> k$, т.е. предполагается, что имеется много относительно коротких рабочих модулей и с определенной степенью точности можно располагать модули контроля и модули-буфера в любой точке временного отрезка [0, n]. В дальнейшем мы не подробно останавливаться на этом вопросе.

При оптимальной расстановке для математического ожидания времени выполнения всех модулей получаем

$$M_{опт}(k,k,n) = n + \alpha \cdot k(n/k)^2 = n + \alpha n^2 / k.$$

Если модуль-буфер только один, то

$$MA(k,1,n) = (1 - a \cdot n)n + (a \cdot x_1)(n + x_1) + ...$$
$$+ a \cdot x_{k-1}(S + \sum_{i=1}^{k-1} x_i) + a \cdot x_k (n + n) \quad ,$$

или после преобразований

$$MA(k,1,n) = n + a \sum_{i=1}^{k} \sum_{j=i}^{k} x_i x_j. \quad (2)$$

Следующая теорема показывает, что для того, чтобы минимизировать данное выражение, модули контроля, как и в предыдущем случае, надо расставлять равномерно.

Теорема 4. В случае m = 1 в оптимальной расстановке k модулей контроля делят отрезок [0, n] на k равных частей. При этом

$$M_{опт}(k,1,n) = n + a \cdot n^2 \frac{(k+1)}{2k}. \qquad (3)$$

4.3.3. Случай $m = 2$

В этом разделе мы рассмотрим случай, когда имеется цепочка из n последовательных рабочих модулей $M_1, M_2,\ldots, M_n$, k модулей контроля и два модуля-буфера. Предположим, что модули контроля $C_1, C_2,\ldots, C_k$ и модули-буфера B_1, B_2 уже расставлены оптимальным образом. Обозначим эту расстановку $A(k,2,n)$. Тогда модуль B_1 располагается перед первым рабочим модулем, а модуль B_2, согласно теореме 2, совпадает с некоторым модулем контроля. Пусть это модуль контроля C_x. Тогда из теорем 3, 4 следует, что первые x и последние $y = k - x$ модулей контроля расположены равномерно. Пусть модуль-буфер B_2 расположен между рабочими модулями M_t и M_{t+1}. Тогда согласно (3) имеем

$$MA(k,2,n) = n + a \cdot t^2 \frac{x+1}{2x} + a \cdot (n-t)^2 \frac{y+1}{2y}. \qquad (4)$$

Согласно предположению 2 разд. 4.3 можно считать, что модуль-буфер B_2 может быть расположен в любой точке t временного отрезка $[0, n]$. Поскольку расстановка $A(k,2,n)$ оптимальна, то при изменении t величина $MA(k,2,n)$ должна только увеличиваться. Следовательно

$$\frac{\partial MA(k,2,n)}{\partial t} = a \cdot 2t \frac{x+1}{2x} - a \cdot 2(n-t) \frac{y+1}{2y} = 0.$$

Отсюда

$$\begin{aligned} t &= n \cdot \frac{y+1}{2y} : \left(\frac{x+1}{2x} + \frac{y+1}{2y} \right) = n \cdot \frac{x(y+1)}{2xy + x + y}; \\ n - t &= n \cdot \frac{y(x+1)}{2xy + x + y}. \end{aligned} \qquad (5)$$

Подставляя полученные в (5) значения t и n-t в (4), получаем:

$$MA(k,2,n)-n=a\cdot n^2\cdot\left(\frac{x(y+1)}{2xy+x+y}\right)^2\cdot\frac{x+1}{2x}+$$
$$+a\cdot n^2\cdot\left(\frac{y(x+1)}{2xy+x+y}\right)^2\cdot\frac{y+1}{2y}=$$
$$=\frac{a\cdot n^2}{2(2xy+x+y)}\cdot\frac{x(y+1)^2(x+1)+y(x+1)^2(y+1)}{2xy+x+y}= \qquad (6)$$
$$=a\cdot n^2\frac{(x+1)(y+1)}{2(2xy+x+y)}.$$

Отметим, что величина $x+y=k$ фиксирована. Докажем, что чем меньше разность $|x-y|$, тем меньше величина $MA(k,2,n)$. Действительно, пусть $z=x-y$, тогда $x=\frac{k+z}{2}$, $y=\frac{k-z}{2}$ и

$$\frac{MA(k,2)-n}{a\cdot n^2}=\frac{(x+1)(y+1)}{2(2xy+x+y)}=\frac{xy+x+y+1}{4xy+2(x+y)}=$$
$$=\frac{\frac{1}{4}(k^2-z^2)+k+1}{k^2-z^2+2\cdot k}=\frac{1}{4}\left(1+\frac{2\cdot k+1}{k^2+2\cdot k-z^2}\right),$$

что и доказывает утверждение. Следовательно, при четном *k* модуль-буфер следует расположить вместе со средним модулем контроля (положить $x=y=k/2$), при нечетном $k\,(k=2\cdot b+1)$ положить, например, $x=b$, $y=b+1$. Затем по формулам (5) вычислить *t* и *n-t* (т. е. определить место расположения модулей-буферов) и модулями контроля разбить каждый из полученных отрезков на равные части. В следующей теореме мы получим выражение для математического ожидания времени выполнения всех модулей в оптимальной расстановке.

Теорема 5. Пусть имеется цепочка из *n* последовательных рабочих модулей. Пусть *k* модулей контроля и два модуля-буфера расположены оптимальным образом. Тогда математическое ожидание времени выполнения всех модулей задается формулой

$$M_{опт}(k,2,n)=\begin{cases} n+\dfrac{a\cdot n^2}{4}+\dfrac{a\cdot n^2}{2k}, & \textit{если } k \textit{ четное},\\ n+\dfrac{a\cdot n^2}{4}+\dfrac{a\cdot n^2}{2k}+ & \\ \dfrac{a\cdot n^2}{2k(k^2+2k-1)}, & \textit{если } k \textit{ нечетное}. \end{cases}$$

Пример 1. При k=3, m=2 получаем, что $b=1$, $x=1$, $y=2$, $t=\frac{3}{7}n$, $n-t=\frac{4}{7}n$. Оптимальная расстановка модулей контроля и модулей-буферов следующая. Модуль-буфер B_1 располагаем вначале цепочки всех модулей, а B_2 – после $\left[\frac{3}{7}\cdot n\right]$ рабочих модулей. Модуль контроля C_1 располагается вместе с модулем B_2, C_2 после $\left[\frac{5}{7}\cdot n\right]$ рабочих модулей, а C_3 – в конце цепочки модулей. Если n делится на 7, то такая расстановка будет оптимальной. Если же нет, то поскольку $n >> k$, она будет близка к оптимальной. Согласно теореме 5 для оптимальной расстановки в данном случае имеем

$$M_{опт}(3,2,n)=n+\frac{a\cdot n^2}{4}+\frac{a\cdot n^2}{2\cdot 3}+\frac{a\cdot n^2}{2\cdot 3(3^2+2\cdot 3-1)}=n+\frac{3}{7}\cdot a\cdot n^2.$$

4.3.4. Случай произвольного *m*

В этом разделе мы построим оптимальную расстановку для общего случая, при условии выполнения предположений 1 – 3 разд. 4.3. Имеется цепочка из n последовательных рабочих модулей M_1, M_2,…, M_n, k модулей контроля $C_1, C_2, \ldots, C_k$ и m модулей-буферов B_1, B_2,…, B_m. Нужно расставить эти модули контроля и модули-буфера между рабочими модулями оптимальным образом. По теореме 2 в оптимальной расстановке расположение каждого модуля-буфера (кроме первого) совпадает с расположением некоторого модуля контроля. Пусть в некоторой расстановке *A(k,m,n)* модуль-буфер B_1 расположен вначале, расположение модуля-буфера B_2 совпадает с расположением модуля контроля C_{b_1}, модуля-буфера B_3 – с модулем контроля $C_{b_1+b_2}$, …, мо-

дуля-буфера B_m – с модулем контроля $C_{b_1+b_2+...+b_{m-1}}$. Определим также $b_m = k - \sum_{i=1}^{m-1} b_i$. Пусть модули-буфера разбивают цепочку из n рабочих модулей на отрезки I_1, I_2,…,I_m, в которых соответственно t_1, t_2,…, t_m рабочих модулей ($\sum_{i=1}^{m} t_i = n$). Согласно теореме 3, в оптимальной расстановке на каждом из отрезков I_i модули контроля должны быть расположены оптимальным образом. Тогда из теоремы 4 и выражения (3) получаем:

$$MA(k,m,n) = \sum_{i=1}^{m}\left(t_i + a \cdot t_i^2 \frac{(b_i+1)}{2 \cdot b_i}\right) = n + a\sum_{i=1}^{m} t_i^2 \frac{(b_i+1)}{2 \cdot b_i}.$$

Отсюда следует, что отрезки I_i вместе с расположенными на них модулями контроля можно произвольно менять местами (поскольку в выражении для $MA(k,m,n)$ при этом только поменяются местами некоторые слагаемые).

Теорема 6. В оптимальной расстановке $\max b_i - \min b_i \le 1$.

Из теоремы 6 следует, что для построения оптимальной расстановки можно представить k в виде $k = m \cdot b + r$, $r < m$ и положить $b_1 = b_2 = ... = b_r = b+1$, $b_{r+1} = ... = b_m = b$. Пусть $q = m - r$.

Лемма 1. 1) $t_1 = t_2 = ... = t_r$. 2) $t_{r+1} = t_{r+2} = ... = t_m$.

Из теоремы 4 следует, что на каждом отрезке I_i модули контроля надо расставить равномерно. Будем искать оптимальную расстановку среди расстановок *A(k,m,n)*, удовлетворяющих этому условию, а также ограничениям, следующим из теоремы 5 и леммы 1. При такой расстановке модуль-буфер B_{r+1} делит n рабочих модулей на две группы размерами $r \cdot t_1$ и $q \cdot t_2$ соответственно. Из теоремы 4 согласно (3) получаем:

$$MA(k,m,n) = r\left(t_1 + a \cdot t_1^2 \frac{x+1}{2x}\right) + q\left(t_m + a \cdot t_m^2 \frac{y+1}{2y}\right), \quad (7)$$

где $x = b+1$, $y = b$.

Будем искать минимум данного выражения по t_1 и t_m, при условии $r \cdot t_1 + q \cdot t_m = n$. При поиске минимума, как обычно, будем рассматри-

вать эти числа как действительные. Предположим сначала, что $r \neq 0$. Для функции Лагранжа

$$v = n + rt_1^2 \frac{x+1}{2x} \cdot a + qt_m^2 \frac{y+1}{2y} \cdot a + \lambda(rt_1 + qt_m - n)$$

имеем систему уравнений

$$\begin{cases} 2rt_1 \dfrac{x+1}{2x} \cdot a + \lambda r = 0, \\ 2qt_m \dfrac{y+1}{2y} \cdot a + \lambda q = 0, \\ rt_1 + qt_m = n, \end{cases}$$

откуда $t_1 \dfrac{x+1}{2x} = t_m \dfrac{y+1}{2y} = -\dfrac{\lambda}{2 \cdot a}$. Обозначим $f(x) = \dfrac{x+1}{2x}$, тогда $t_1 = \dfrac{f(y)}{f(x)} \cdot t_m$. Подставляя это значение t_1 в третье уравнение системы, получим

$$r \frac{f(y)}{f(x)} \cdot t_m + qt_m = n, \quad t_m = \frac{n}{r \cdot \dfrac{f(y)}{f(x)} + q} = \frac{f(x) \cdot n}{rf(y) + qf(x)},$$

$$\begin{cases} t_1 = \dfrac{f(b)}{rf(b) + qf(b+1)} \cdot n, \\ t_m = \dfrac{f(b+1)}{rf(b) + qf(b+1)} \cdot n, \end{cases} \qquad (8)$$

где $f(x) = \dfrac{x+1}{2x}, k = mb + r, q = m - r$.

Если $r = 0$, то t_1 теряет смысл, $t_2 = n/m$ и это же значение получается при подстановке $r = 0$ в (8). Таким образом, формулы (8) верны при всех r.

Опишем алгоритм построения оптимальной расстановки модулей контроля и модулей-буферов. По заданным числам k и m находим b, равное частному от деления k на m, остаток r, и число $q = m - r$. По формулам (8) вычисляем числа t_1 и t_m. Модули-буфера B_1, B_2, …, B_r располагаем с интервалами, равными t_1, а B_{r+1}, B_{r+2}, …, B_m – с интервалами, равными t_m. В каждом из полученных отрезков I_1, I_2,…, I_r равномерно располагаем $b+1$ модуль контроля, а в каждом из отрезков I_{r+1},

$I_{r+2},\ldots, I_m$ равномерно располагаем b модулей контроля. В следующей теореме мы получим выражение для математического ожидания времени выполнения всех модулей в оптимальной расстановке.

Теорема 7. Пусть имеется цепочка из n последовательных рабочих модулей. Пусть k модулей контроля и m модулей-буферов расположены оптимальным образом (при этом k и m произвольные натуральные числа, не обязательно $k \geq m$). Тогда математическое ожидание времени выполнения всех модулей задается формулой

$$M_{опт}(k,m,n) = n + \frac{a \cdot n^2}{2m} + \frac{a \cdot n^2}{2k} + a \cdot n^2 \frac{r(m-r)}{2km(m(b^2+2b)+r)}, \qquad (9)$$

где $k = m \cdot b + r,\ r < m$.

Рассмотрим пример реализации описанного алгоритма.

Пример 2. При $k=4$, $m=3$ получаем, что $b = r = 1$, $q = 2$.

$$f(b) = \frac{b+1}{2 \cdot b} = 1,\ f(b+1) = f(2) = \frac{3}{4}.$$

По формулам (8) получаем:

$$t_1 = \frac{f(b)}{rf(b) + qf(b+1)} \cdot n = \frac{1}{1 + 2 \cdot (3/4)} = 0.4 \cdot n,$$

$$t_2 = t_3 = \frac{3/4}{1 + 2 \cdot (3/4)} \cdot n = 0.3 \cdot n.$$

Оптимальная расстановка модулей контроля и модулей-буферов следующая. Модуль-буфер B_1 располагаем вначале цепочки всех модулей, B_2 – после $[0.4 \cdot n]$ рабочих модулей, а B_3 – после $[0.7 \cdot n]$ рабочих модулей. Модуль контроля C_1 располагается после $[0.2 \cdot n]$ рабочих модулей, модули контроля C_2 и C_3 располагаются вместе с модулями-буферами B_2 и B_3 соответственно, а модуль C_4 - в конце цепочки модулей. Согласно теореме 7, для оптимальной расстановки в данном случае получаем

$$M_{опт}(4,3,n) = n + \frac{a \cdot n^2}{2 \cdot 3} + \frac{a \cdot n^2}{2 \cdot 4} + a \cdot n^2 \frac{1(3-1)}{2 \cdot 3 \cdot 4(3(1^2 + 2 \cdot 1) + 1)} =$$

$$= n + 0.3 \cdot a \cdot n^2.$$

Как видим, математическое ожидание получилось меньше, чем в примере 1. Это объясняется тем, что были добавлены один модуль контроля и один модуль-буфер.

4.4. Расположение модулей контроля для случая нескольких параллельных цепочек рабочих модулей

В настоящем разделе рассмотрен случай, когда граф зависимости по данным состоит из нескольких параллельных цепочек. Из условия допустимости расстановки следует, что в начале каждой цепочки рабочих модулей необходимо расположить модуль-буфер, а в конце каждой цепочки необходим модуль контроля. Это значит, что при $k < l$ или $m < l$ допустимой расстановки не существует. В дальнейшем мы будем предполагать, что $k \geq l$, $m \geq l$, и рассматривать только допустимые расстановки.

Поскольку $n_i >> k$, $a << \frac{1}{n_i}$ $i = 1,...,l$, т.е. в каждой цепочке достаточно много рабочих модулей, а ошибка при их выполнении возможна, но маловероятна, то для оптимальной расстановки модулей контроля и модулей-буферов внутри каждой цепочки можно использовать алгоритм, описанный в разд. 4.3. Следовательно, надо только решить задачу оптимального распределения модулей контроля и модулей-буферов между цепочками рабочих модулей.

Пусть на *i*-ю цепочку рабочих модулей выделено k_i модулей контроля и m_i модулей-буферов. Произвольную допустимую расстановку этих модулей внутри данной цепочки будем обозначать через $A'(k_i, m_i, n_i)$, а оптимальную расстановку – через $A'_{опт}(k_i, m_i, n_i)$. Через $MA'(k_i, m_i, n_i)$ (и соответственно $M'_{опт}(k_i, m_i, n_i)$) обозначим математическое ожидание суммарной длительности выполнения всех рабочих модулей данной цепочки (включая их повторное выполнение при возникновении ошибок). Общую допустимую расстановку k модулей контроля и m модулей-буферов по всей совокупности цепочек рабочих модулей будем обозначать $A(k, m, n)$, а оптимальную расстановку – $A_{опт}(k, m, n)$. Соответствующие им математические ожидания

суммарной длительности выполнения всех модулей графа G будем обозначать $MA(k,m,n)$ и $M_{опт}(k,m,n)$.

Важно отметить, что, несмотря на появление независимых цепочек рабочих модулей, которые в принципе можно было бы выполнять параллельно, мы будем предполагать, что имеется только один процессор, который выполняет рабочие модули. В этом случае время, необходимое для выполнения всех рабочих модулей, равняется сумме времен для каждой из цепочек, т.е.

$$MA(k,m,n) = \sum_{i=1}^{l} MA'(k_i,m_i,n_i).$$

Из этой формулы, в частности, следует, что в общей оптимальной расстановке $A_{опт}(k,m,n)$ расстановка внутри каждой цепочки тоже должна быть оптимальной. Кроме того, мы должны так распределить k модулей контроля и m модулей-буферов по l цепочкам, чтобы минимизировать $MA(k,m,n)$. Следовательно, для оптимальной расстановки получаем следующее выражение:

$$M_{опт}(k,m,n) = \min_{k_i,m_i:\sum k_i=k,\sum m_i=m} \sum_{i=1}^{l} M'_{опт}(k_i,m_i,n_i). \qquad (10)$$

В последующих разделах мы рассмотрим различные подходы к решению этой задачи.

4.4.1. Разделение модулей контроля по одинаковым цепочкам

В этом разделе мы рассмотрим частный случай поставленной задачи. Пусть имеется l одинаковых цепочек, в каждой из которых – по s рабочих модулей ($n = ls$), и на каждую из цепочек выделено по w модулей-буферов ($m = lw$). Нужно оптимально распределить k модулей контроля по l цепочкам.

Предположим сначала, что $m = 1$, т.е. $w = 1$. В этом случае в начале каждой цепочки находится модуль-буфер и других модулей-буферов нет. Из результатов, полученных в [135], следует, что в этом случае внутри каждой цепочки модули контроля следует располагать равномерно, а минимальное математическое ожидание времени выполнения задается формулой

$$M'_{опт}(k_i,1,s) = s + a \cdot s^2 \frac{k_i + 1}{2k_i}.$$

В этом случае

$$M_{опт}(k,m,n) = \min_{k_i:\sum k_i = k} \sum_{i=1}^{l} M'_{опт}(k_i,1,s) = n + l \cdot a \cdot s^2 \min_{k_i:\sum k_i = k} \sum_{i=1}^{l} \frac{k_i + 1}{2k_i}. \quad (11)$$

Рассмотрим функцию $f(x) = \frac{x+1}{2x}$. Несложно заметить, что эта функция является строго выпуклой при $x > 0$. Действительно, $f'(x) = -0{,}5x^{-2}$, $f''(x) = x^{-3} > 0$ при $x > 0$.

Лемма 2. Пусть $f(x)$ – строго выпуклая функция. Рассмотрим задачу оптимизации $\min_{k_i:\sum k_i = k} \sum_{i=1}^{l} f(k_i)$, где $k_1, k_2, ..., k_l$ – натуральные числа. Тогда для оптимального набора $k_1, k_2, ..., k_l$ выполнено неравенство $\max_i k_i - \min_i k_i \le 1$.

Если $f(x)$ – нестрого выпуклая функция, то таким свойством обладает как минимум один из оптимальных наборов $k_1, k_2, ..., k_l$.

Из леммы 2 и строгой выпуклости функции $f(x) = \frac{x+1}{2x}$ следует, что минимум в выражении (11) достигается, если модули контроля распределять равномерно между цепочками рабочих модулей.

Рассмотрим теперь случай, когда w – произвольное. Тогда выражение (10) можно записать в виде

$$M_{опт}(k,m,n) = \min_{k_i:\sum k_i = k} \sum_{i=1}^{l} M'_{опт}(k_i, w, s), \quad (12)$$

где $M^i_{опт}(k_i, w, s)$ можно вычислить по формулам, приводимым в разд. 4.3:

$$M_{опт}(k,m,n) = n + \frac{a \cdot n^2}{2m} + \frac{a \cdot n^2}{2k} + a \cdot n^2 \frac{r(m-r)}{2km(m(b^2+2b)+r)}. \quad (13)$$

Лемма 3. Для любых натуральных k, h, m, n $(k > h)$ выполняется неравенство $M'_{опт}(k-h,m,n) + M'_{опт}(k+h,m,n) \ge 2M'_{опт}(k,m,n)$.

Из лемм 2, 3 следует теорема.

Теорема 8. Пусть имеется l одинаковых цепочек, каждая из которых состоит из s рабочих модулей ($n = ls$), и на каждую из них выделено по w модулей-буферов ($m = lw$). Пусть $k = l \cdot \bar{b} + \bar{r}$, $\bar{r} < l$. Тогда математическое ожидание времени выполнения всех модулей задается формулой

$$M_{опт}(k,m,n) = \bar{r}M'_{опт}(\bar{b}+1,w,s) + (l-\bar{r})M'_{опт}(\bar{b},w,s),$$

где $M'_{опт}(\bar{b}+1,w,s)$ и $M'_{опт}(\bar{b},w,s)$ задаются формулой (13).

Из теоремы 8 следует способ построения оптимальной расстановки. Для этого следует назначить по $\bar{b}+1$ модулей контроля на некоторые $\bar{r}$ цепочек, по $\bar{b}$ модулей контроля – на остальные $l-\bar{r}$ цепочек и применить алгоритм оптимальной расстановки для каждой цепочки.

4.4.2. Распределение модулей контроля по цепочкам при заданном распределении модулей-буферов

Как и в предыдущем разделе, будем предполагать, что m модулей-буферов уже некоторым образом распределены по l цепочкам (m_i модулей-буферов в i-й цепочке). В предыдущем разделе мы доказали, что если $n_1 = n_2 = ... = n_l$ и $m_1 = m_2 = ... = m_l$, то модули контроля также следует располагать между цепочками равномерно. В этом разделе числа $n_1, n_2, ..., n_l$ и $m_1, m_2, ..., m_l$ не предполагаются равными.

В данном случае выражение (10) можно записать в виде

$$M_{опт}(k,m,n) = \min_{k_i, m_i: \sum k_i = k} \sum_{i=1}^{l} M'_{опт}(k_i, m_i, n_i).$$

Поскольку величины $n_1, n_2, ..., n_l$ и $m_1, m_2, ..., m_l$ фиксированы, $M'_{опт}(k_i, m_i, n_i)$ можно обозначить как $q_i(k_i)$, и задача сводится к нахождению $\min_{k_i, \sum k_i = k} \sum_{i=1}^{l} q_i(k_i)$. В следующей лемме мы сформулируем алгоритм для решения этой задачи.

Лемма 4. Пусть $q_i(a)$ ($a \in N, i = 1,...,l$) – функции натурального аргумента, такие, что для каждой из них

$$q_i(a) - q_i(a+1) \le q_i(a-1) - q_i(a) \qquad (14)$$

при всех $a \geq 2$. Тогда для нахождения величины $\min_{k_i:\sum k_i=k} \sum_{i=1}^{l} q_i(k_i)$ существует алгоритм сложности не более $Cl^2 \ln^2 k$.

Доказательство. Для каждой пары натуральных a и i введем следующие обозначения: $r(a,i) = q_i(a) - q_i(a+1)$, $Q = \{(j,i) : j \leq k_i\}$. Тогда $q_i(k_i) = q_i(1) - \sum_{j=1}^{k_i} r(j,i)$, $\sum_{i=1}^{l} q_i(k_i) = \sum_{i=1}^{l} q_i(1) - \sum_{(j,i)\in Q} r(j,i)$. Поскольку величина $\sum_{i=1}^{l} q_i(1)$ постоянна, то надо решить задачу $\max_{Q} \sum_{(i,j)\in Q} r(j,i)$. В последнем выражении содержится k слагаемых, так как $\sum_{i=1}^{l} k_i = k$. Из (14) следует, что

$$r(j,i) \geq r(j+1,i) \tag{15}$$

при всех i, j. Это означает, что последняя задача эквивалентна задаче нахождения k максимальных чисел среди $r(j,i), j = 1,...,k, i = 1,...,l$, которая легко решается за время $P(l,k)$. Мы решим ее за время $P(l, \ln k)$, используя условие (15). Найдем величину k_l. Для этого определим функцию $U(c)$, $c = 1,...,k$, равную количеству таких пар (j,i), что $r(j,i) \leq r(c,l)$. Заметим, что при фиксированном i количество значений j таких, что $r(j,i) \leq B$ ($B = const$), определяется за время $C \ln k$, а значит, значение $U(c)$ вычисляется за время $Cl \ln k$. Поскольку $r(k_l,l)$ входит в множество k максимальных среди $r(j,i)$ чисел, то количество чисел, не меньших $r(k_l,l)$, не превышает k, т.е. $U(k_l) \leq k$. С другой стороны, $r(k_l+1,l)$ уже не входит в множество k максимальных среди $r(j,i)$ чисел, поэтому $U(k_l+1) > k$. Таким образом, k_l можно найти, вычисляя $C \ln k$ раз функцию U. Вычисление одного значения функции U требует $Cl \ln k$ операций, поэтому общее число операций для нахождения числа k_l составляет $Cl \ln^2 k$, а для нахождения всех чисел $k_1, k_2, ..., k_l$ требуется $Cl^2 \ln^2 k$ операций. Лемма доказана.

Из леммы 4 следует выполнение условия (13) для функций $q_i(k_i) = M'_{опт}(k_i, m_i, n_i)$ и алгоритм решения поставленной в этом разделе задачи.

4.4.3. Приближенный алгоритм расстановки модулей

В предыдущих разделах предполагалось, что *m* модулей-буферов некоторым образом распределены по *l* цепочкам рабочих модулей, и были разработаны алгоритмы для распределения по этим цепочкам модулей контроля. В этом разделе мы рассмотрим общий случай сформулированной задачи: требуется распределить *k* модулей контроля и *m* модулей-буферов по *l* цепочкам (k_i модулей контроля и m_i модулей-буферов на i-ю цепочку, $\sum k_i = k$, $\sum m_i = m$) так, чтобы минимизировать величину $MA(k, m, n) = \sum_{i=1}^{l} MA'(k_i, m_i, n_i)$. Будем предполагать, что $k \geq m$. В этом разделе мы рассмотрим приближенный алгоритм решения этой задачи.

Лемма 5. Для случая одной цепочки при $k \geq m$

$$M'_{опт}(k, m, n) = n + \frac{a \cdot n^2}{2m} + \frac{a \cdot n^2}{2k} + \delta(k, m, n),$$

где $\delta(k, m, n) < 0{,}03\left(\frac{a \cdot n^2}{2m} + \frac{a \cdot n^2}{2k}\right)$.

Из леммы 5 следует, что $\frac{M'_{опт}(k, m, n) - n}{\alpha} \approx \frac{n^2}{2m} + \frac{n^2}{2k}$. Поэтому вместо исходной мы будем рассматривать следующую задачу: найти такие целые неотрицательные числа $m_1, m_2, ..., m_l$, $k_1, k_2, ..., k_l$ ($\sum_{i=1}^{l} m_i = m$, $\sum_{i=1}^{l} k_i = k$), при которых величина $\sum_{i=1}^{l}\left(\frac{n_i^2}{2m_i} + \frac{n_i^2}{2k_i}\right)$ минимальна. Преимущество данного приближения в том, что можно отдельно решать задачи $\min_{m_i} \sum_{i=1}^{l} \frac{n_i^2}{2m_i}$ и $\min_{k_i} \sum_{i=1}^{l} \frac{n_i^2}{2k_i}$.

Рассмотрим набор функций $q_i(a) = \frac{n_i^2}{2a}$. Покажем, что для каждой из них выполнено условие (14). Это следует из приводимых ниже эквивалентных неравенств:

$$-q(a+1,b) + q(a,b) \le -q(a,b) + q(a-1,b);$$

$$-\frac{n_i^2}{2(a+1)} + \frac{n_i^2}{2a} \le -\frac{n_i^2}{2a} + \frac{n_i^2}{2(a-1)};$$

$$-\frac{1}{a+1} + \frac{1}{a} \le -\frac{1}{a} + \frac{1}{a-1}; \frac{1}{a(a+1)} \le \frac{1}{a(a-1)}.$$

Применяя лемму 4 для функций $q_i(a)$ и учитывая ограничение $\sum_{i=1}^{l} m_i = m$, получаем числа $m_1, m_2, ..., m_l$, а учитывая ограничение $\sum_{i=1}^{l} k_i = k$, получаем числа $k_1, k_2, ..., k_l$. Из условия $m \le k$ и леммы 1 следует, что $m_i \le k_i$ при всех $i = 1,...,l$. После этого для i-й цепочки ($i = 1,2,...,l$) следует расставить m_i модулей-буферов и k_i модулей контроля согласно алгоритму для случая одной цепочки.

Оценим относительную погрешность данного алгоритма. В результате его работы мы получаем расстановку $A(k,m,n)$, для которой

$$MA(k,m,n) = \sum_{i=1}^{l} M_{onm}(k_i, m_i, n_i) = n +$$

$$+ \sum_{i=1}^{l} \left(\frac{a \cdot n_i^2}{2m_i} + \frac{a \cdot n_i^2}{2k_i} + \delta(k_i, m_i, n_i) \right).$$

Пусть в оптимальной расстановке в i-й цепочке назначено m_i^* модулей-буферов и k_i^* модулей контроля. Тогда

$$M_{onm}(k,m,n) = n + \sum_{i=1}^{l} \left(\frac{a \cdot n_i^2}{2m_i^*} + \frac{a \cdot n_i^2}{2k_i^*} + \delta(k_i^*, m_i^*, n_i) \right).$$

Из леммы 4 следует, что $\delta(k_i, m_i, n_i) < 0{,}03 \left(\frac{a \cdot n_i^2}{2m_i} + \frac{a \cdot n_i^2}{2k_i} \right)$. Суммируя, получаем $\sum_{i=1}^{l} \delta(k_i, m_i, n_i) < 0{,}03 \sum_{i=1}^{l} \left(\frac{a \cdot n_i^2}{2m_i} + \frac{a \cdot n_i^2}{2k_i} \right)$. Отсюда следует,

что $\frac{M(k,m.n)-n}{\alpha} \le 1{,}03\sum_{i=1}^{l}\left(\frac{n_i^2}{2m_i}+\frac{n_i^2}{2k_i}\right)$. Но согласно выбору m_i и k_i имеем

$$\sum_{i=1}^{l}\left(\frac{n_i^2}{2m_i}+\frac{n_i^2}{2k_i}\right) \le \sum_{i=1}^{l}\left(\frac{n_i^2}{2m_i^*}+\frac{n_i^2}{2k_i^*}\right) \le$$

$$\le \sum_{i=1}^{l}\left(\frac{n_i^2}{2m_i^*}+\frac{n_i^2}{2k_i^*}+\delta(m_i^*,k_i^*,n_i)\right)=\frac{M_{onm}(k,m,n)-n}{\alpha}.$$

Следовательно, при применении описанного выше приближенного алгоритма математическое ожидание количества рабочих модулей, которые придется выполнить повторно, может увеличиться не более чем на 3%. Оценка сложности приведенного алгоритма следует из леммы 4 и составляет $O(Cl^2 \ln^2 k)$.

4.5. Расположение модулей контроля для случая ориентированного дерева

4.5.1. Постановка задачи

В этом разделе мы продолжим разрабатывать алгоритмы для оптимальной организации подсистемы контроля в системах реального времени. В разд. 4.3 мы разработали алгоритм построения оптимальной расстановки модулей контроля и модулей-буферов для случая, когда граф *G* зависимости по данным между рабочими модулями является цепочкой из *n* вершин. В разд. 4.4 был рассмотрен случай, когда граф *G* состоит из нескольких цепочек, не связанных между собой. В данном разделе мы рассмотрим случай, когда граф *G*=<*V,E*> является ориентированным деревом, дуги ориентированы от листьев к корню снизу вверх. В этом случае для каждой вершины $i \in V$, кроме корня, существует ровно одна дуга $(i, j) \in E$.

Этот случай имеет большое практическое значение. Он соответствует системе рабочих модулей, которые рассчитывают один результат (который выдает рабочий модуль, являющийся корнем дерева). Для расчета этого результата этот рабочий модуль требует несколько вспомогательных результатов, для расчета каждого из этих вспомога-

тельных результатов снова нужны предварительные расчеты и т. д. При этом все промежуточные результаты используются ровно один раз, и возникает структура, описываемая деревом.

В этом разделе мы отказываемся от упрощающих предположений 1-2 разд. 4.3, но по-прежнему предполагаем, что выполняется упрощение 3, т.е. ошибка при выполнении рабочих модулей возможна, но маловероятна. Кроме того, по-прежнему, предполагается, что имеется только один процессор, который производит вычисления.

Из условия допустимости расстановки, сформулированного в разд. 4.2, следует, что после рабочего модуля, который является корнем дерева, необходимо расположить модуль контроля, а перед рабочими модулями, которые являются листьями дерева, необходимо расположить модули-буфера. Мы, однако, в этом разделе сделаем более сильное предположение. Будем считать, что количество модулей-буферов равняется количеству рабочих модулей ($m = n$), и мы можем расположить модуль-буфер перед каждым рабочим модулем.

Обоснование такого предположения следующее. Пусть A – произвольная вершина (рабочий модуль), в которую входит более одного ребра. Другими словами, в графе G существуют ребра BA и CA, где B и C некоторые рабочие модули. Предположим, что перед рабочим модулем A нет модуля-буфера. Пусть, например, по расписанию сначала выполняется модуль B. Тогда после него не может сразу выполниться модуль A, так как перед этим надо выполнить C. Но во время выполнения C потеряются выходные данные B. Получили противоречие. Следовательно, надо предположить, что существует способ как-то сохранить выходные данные B, пока выполняется модуль C. Но в этом случае эти сохраненные данные можно использовать и при перезапуске системы. А это, согласно определению, эквивалентно предположению, что перед перед рабочим модулем A располагается модуль-буфер.

В работе Белого и Сушкова [134], которая стала основой для нашей модели, не предполагается наличие модуля-буфера перед каждым рабочим модулем. Там, однако, рассматривался случай многопроцессорной системы, при которой возможна ситуация, когда рабочие модули B и C заканчивают выполнение одновременно, и тогда описан-

ной проблемы сохранения промежуточных результатов не возникает. Мы же во всех предыдущих разделах рассматривали только графы, для которых как входящая, так и исходящая степень не превосходила единицы, и поэтому предположение о наличии модуля-буфера перед каждым рабочим модулем тоже не требовалось.

Таким образом, в этом разделе мы рассмотрим задачу оптимального расположения k модулей контроля в дереве из n рабочих модулей, перед каждым из которых расположен модуль-буфер. Нужно расположить модули контроля так, чтобы математическое ожидание времени, затраченного на выполнение всех рабочих модулей, было минимальным. Будем предполагать, что каждый модуль контроля располагается непосредственно после некоторого рабочего модуля. Произвольную допустимую расстановку модулей контроля будем обозначать, как обычно, через $A(k,n,n)$, оптимальную – через $A_{опт}(k,n,n)$, а соответствующие им математические ожидания длительности выполнения всех модулей $MA(k,n,n)$ и $M_{опт}(k,n,n)$. Обозначим также через $A'(k,k,n)$ (и соответственно $M'(k,k,n)$) расстановку k модулей контроля и k модулей-буферов в цепочке из n рабочих модулей (и соответствующее математическое ожидание времени выполнения). В следующем разделе мы установим взаимосвязь между этими расстановками.

4.5.2. Связь расстановок модулей контроля в дереве и в цепочке

Рассмотрим произвольную расстановку модулей контроля $A(k,n,n)$. Будем предполагать, что модули контроля занумерованы от 1 до k в соответствии с порядком, задаваемым деревом G (т.е. если i-й модуль контроля является последователем j-го, то $i \geq j$). Поставим каждому рабочему модулю в соответствие модуль контроля, являющийся его ближайшем последователем в дереве G. Пусть i-й модуль контроля (i=1, 2, …, k) поставлен в соответствие a_i рабочим модулям. Тогда $\sum_{i=1}^{k} a_i = n$.

Рассмотрим теперь расстановку $A'(k,k,n)$ k модулей контроля в цепочке из n рабочих модулей такую, что модули контроля совпадают с модулями-буферами и разбивают рабочие модули на k групп, по $a_1, a_2, ..., a_k$ модулей в каждой группе. Следующая лемма связывает эти две расстановки.

Лемма 6. $MA(k,n,n) = MA'(k,k,n) = n + a\sum_{i=1}^{k} a_i^2$.

В разд. 4.3 мы установили, что в оптимальной расстановке для случая цепочки все числа $a_1, a_2, ..., a_k$ равны между собой. Однако, в рассматриваемом случае мы не всегда можем разбить дерево на k равных частей. Следовательно, задача сводится к тому, чтобы разбить модулями контроля дерево на k частей (с количеством вершин $a_1, a_2, ..., a_k$) с минимальной величиной $\sum_{i=1}^{k} a_i^2$, т.е. в некотором смысле на k "как можно более равных" частей. В следующем разделе мы построим полиномиальный алгоритм для такого разбиения.

4.5.3. Алгоритм построения оптимальной расстановки модулей контроля в дереве.

Для каждой фиксированной расстановки $A(k,n,n)$ определим величины $a_1, a_2, ..., a_k$ так, как это сделано в разд. 4.5.2. По лемме 6 для этой расстановки $MA(k,n,n) = n + a\sum_{i=1}^{k} a_i^2$. Следовательно, оптимальной будет такая расстановка, для которой величина $\sum_{i=1}^{n} a_i^2$ минимальна.

Теорема 9. Существует полиномиальный алгоритм расстановки модулей контроля для дерева. Его сложность составляет $(lk^3n^4)P(\ln k, \ln n)$, где $l+1$ – максимальная степень вершин в дереве, P – некоторый полином.

4.5.4. Расположение модулей контроля для случая произвольного графа связей между рабочими модулями

В этом разделе мы рассмотрим случай, когда граф зависимости по данным G между рабочими модулями произвольный. Как и в предыдущем разделе, будем предполагать, что процессор один, вероятность возникновения ошибки $a << \frac{1}{n}$ (предположение 3 разд. 4.3), а перед каждым рабочим модулем расположен модуль-буфер. Нужно расставить k модулей контроля ($k < n$, каждый модуль контроля располагается непосредственно после некоторого рабочего модуля), так чтобы минимизировать время работы всей системы.

4.5.4.1. Порядок выполнения рабочих модулей при заданном расположении модулей контроля

Поскольку граф зависимости по данным $G=<V,E>$ больше не является деревом, то, вообще говоря, один и тот же рабочий модуль теперь может контролироваться несколькими независимыми модулями контроля. Это приводит к тому, что даже при фиксированной расстановке модулей контроля математическое ожидание времени работы всей системы может быть различным в зависимости от порядка выполнения рабочих модулей и модулей контроля и даже от стратегии наших действий при обнаружении ошибки.

Рассмотрим простой пример, когда вершинами графа G являются 5 рабочих модулей $A_1, A_2, ..., A_5$ (вместе с модулями-буферами перед каждым из них, которые мы подразумеваем), а ребрами – 4 зависимости до данным (A_1, A_2), (A_2, A_3), (A_3, A_4) и (A_2, A_5). Также имеются два модуля контроля C_1 и C_2. Тогда единственная допустимая расстановка состоит в том, чтобы расположить модули контроля C_1 и C_2 после рабочих модулей A_4 и A_5. Напомним, что согласно предположению 3 разд. 4.3 ошибка при выполнении работ возможна, но маловероятна, а вероятность более чем одной ошибки пренебрежимо мала и даже не рассматривается.

Рассмотрим следующие варианты задания порядка выполнения рабочих модулей и модулей контроля.

1) Выполняем модули A_1, A_2, A_3, A_4, C_1. Если модуль контроля C_1 обнаружил ошибку, повторяем выполнение этих модулей, пока они не выполнятся без ошибки. После этого выполняем A_5 и C_2. При ошибке повторяем выполнение A_5 и C_2.

2) Выполняем модули A_1, A_2, A_5, C_2. Если модуль контроля C_2 обнаружил ошибку, повторяем выполнение этих модулей, пока они не выполнятся без ошибки. После этого выполняем A_3, A_4, C_1, пока C_1 не выполнится без ошибки.

Длительности выполнения рабочих модулей равны 1, модулей контроля – 0. Вероятность возникновения ошибки в каждом из рабочих модулей равна a. Пренебрегая малыми порядка a^2, получим, что при первом порядке выполнения модулей математическое ожидание общего времени выполнения равно

$$M_1 = (1-5a)\cdot 5 + 4a(5+4) + a(5+1) = 5+17a.$$

При втором порядке выполнения модулей аналогично получаем

$$M_2 = (1-5a)\cdot 5 + 3a(5+3) + 2a(5+2) = 5+13a.$$

Таким образом, при изменении порядка выполнения модулей контроля результат поменялся. Следовательно, мы должны найти не только оптимальное расположение модулей контроля, но и оптимальный порядок их выполнения.

Рассмотрим произвольную расстановку модулей контроля $A(k,n,n)$. Пусть известен порядок, в котором они должны выполняться, и пусть они занумерованы в соответствии с этим порядком (от 1 до k). Поставим каждому рабочему модулю в соответствие модуль контроля, являющийся его последователем с наименьшим номером в графе G. Пусть i-й модуль контроля поставлен в соответствие a_i рабочим модулям (i=1, 2, …, k). Тогда $\sum_{i=1}^{k} a_i = n$. Таким образом, расстановка модулей контроля и порядок их выполнения однозначно определяют некоторое разбиение графа G на k подграфов. Будем обозначать такое разбиение графа через R.

Рассмотрим также расстановку $A'(k,k,n)$ k модулей контроля и модулей-буферов в цепочке из n рабочих модулей такую, что модули контроля совпадают с модулями-буферами и разбивают рабочие модули на k групп, по $a_1, a_2, ..., a_k$ модулей в каждой группе. Докажем следующую лемму.

Лемма 7. При заданной расстановке $A(k,n,n)$ и заданном порядке выполнения модулей контроля существует такой допустимый порядок выполнения рабочих модулей, что

$$MA(k,n,n) = MA'(k,k,n) = n + a\sum_{i=1}^{k} a_i^2 .$$

Доказательство. Рассмотрим такой порядок выполнения рабочих модулей. Сначала выполняем a_1 рабочих модулей, соответствующих первому модулю контроля. После этого выполняем первый модуль контроля. Если им будет обнаружена ошибка, то повторим выполнение этих a_1 рабочих модулей, в противном случае выполним a_2 рабочих модулей, соответствующих второму модулю контроля, и т. д. Очевидно, что при таком порядке выполнения искомое математическое ожидание будет таким же, как и в соответствующем случае для цепочки, т.е. равным $MA'(k,k,n)$.

Покажем, что такой порядок выполнения возможен, т. е. не противоречит порядку, задаваемому графом G (при доказательстве леммы 6 для случая дерева это было очевидно). Предположим противное. Пусть при выполнении a_i рабочих модулей, соответствующих некоторому модулю контроля i, необходимо выполнить рабочий модуль M_1, зависящий по данным от рабочего модуля M_2, который выполнить пока нельзя. Это означает, что модулю M_2 соответствует модуль контроля с номером $j > i$. С другой стороны, в графе G модуль контроля i является последователем модуля M_1, который, в свою очередь, является последователем M_2. Следовательно, модуль контроля с номером $i < j$ является последователем M_2, что противоречит выбору модуля контроля j как модуля, соответствующего рабочему модулю M_2. Данное противоречие доказывает возможность описанного выше порядка выполнения рабочих модулей и заканчивает доказательство леммы.

При доказательстве леммы 7 мы описали некоторый порядок выполнения рабочих модулей. Важно отметить, что при обнаружении ошибки модулем контроля, мы сразу же приступали к перезапуску системы. В дальнейшем такую стратегию наших действий будем называть простой стратегией. Такая стратегия выглядит логичной и обычно используется на практике. В разд. 4.5.5 и 4.5.6 при рассмотрении задачи оптимальной расстановки модулей контроля, мы будем предполагать, что при обнаружении ошибки используется именно простая стратегия, и, следовательно, при заданной расстановке $A(k,n,n)$ математическое ожидание $MA(k,n,n)$ определяется леммой 7. В этом случае задачу оптимального расположения модулей контроля можно сформулировать следующим образом.

Задача 1. Пусть задан граф *G* рабочих модулей и k модулей контроля. Определить такую расстановку модулей контроля и порядок их выполнения, что для полученного разбиения *R* графа *G* на k частей (с количеством вершин $a_1, a_2, ..., a_k$) сумма $\sum_{i=1}^{k} a_i^2$ минимальна.

В разд. 4.5.7, однако, мы рассмотрим пример графа *G,* при котором целесообразно применять другую стратегию, т. е. при обнаружении ошибки не приступать к рестарту немедленно.

4.5.5. NP-полнота в сильном смысле задачи расстановки модулей контроля для случая произвольного графа

В этом разделе мы докажем, что задача 1, сформулированная в разд. 4.5.4.1 задача является *NP*-трудной в сильном смысле [51]. Если разбиение *R* разбивает граф *G* на k равных частей ($a_1 = a_2 = ... = a_k$), то сумма $\sum_{i=1}^{k} a_i^2$ будет минимальной. Мы докажем что задача, которая определяет, существует ли такое разбиение *R* (аналог задачи 1 в форме распознавания свойств), является *NP*-полной в сильном смысле.

Рассмотрим *NP*-полную в сильном смысле задачу “3-разбиение”: даны натуральные числа $a_1, a_2, ..., a_{3n}, B$, такие, что

$\frac{B}{4} < a_i < \frac{B}{2}, i = 1,2,...,3n,$ и $\sum_{i=1}^{3n} a_i = nB$. Можно ли разбить множество $\{1,2,\ldots,3n\}$ на непересекающиеся подмножества $A_1, A_2, ...A_n$, такие, что $\sum_{i \in A_j} a_i = B, j = 1,...,n.$ (Очевидно, что если такое разбиение существует, то в каждое A_j входит ровно 3 элемента).

Рассмотрим эту задачу с дополнительным условием $B > C_{3n}^3 - (n+1)$, которое, очевидно, не влияет на *NP*-полноту. Докажем, что задача "3-разбиение" псевдополиномиально сводится к задаче расстановки модулей контроля. Рассмотрим следующий частный случай последней задачи. Пусть в графе G $(B+1)(n+1)$ вершин. Разобьем их на 3 группы. В первой группе *nB* вершин. Они составляют *3n* цепочек (по a_i вершин в каждой цепочке). Во второй группе C_{3n}^3 вершин. Каждой из них взаимно однозначно соответствует тройка цепочек вершин второй группы, причем из последних модулей указанной тройки в эту вершину второй группы ведут дуги графа *G*. В третью группу входят оставшиеся $x = B + n + 1 - C_{3n}^3$ вершин. Эти вершины соединены цепочкой, в первую из вершин которой ведут C_{3n}^3 ребер, по одному из каждой вершины второй группы. Имеется также $n+1$ модулей контроля.

Пусть f – это функция, отображающая индивидуальную задачу I "3-разбиение" в описанную выше индивидуальную задачу расстановки модулей контроля. Определим функции длины и максимума [51]: $l_1 = M_1 = n + B$ в задаче расстановки модулей контроля и $l_2 = n + \ln B$, $M_2 = n + B$ в задаче "3-разбиение. Тогда справедливы следующие утверждения.

1) Функция f вычисляется псевдополиномиальным алгоритмом, т.е. алгоритмом, сложность которого $P(n, B)$, где P – некоторый полином. Доказательство этого факта очевидно.

2) Существует такой полином $g_1(x)$, что $g_1(l_1(f(I))) \geq l_2(I)$, или $g_1(n + B) \geq n + \ln B$. В качестве $g_1(x)$ можно взять *x*.

3) Существует такой полином $g_2(x, y)$, что $g_2(M_2(I), l_2(I)) \geq M_1(f(I))$, или $g_2(n+B, n+\ln B) \geq n+B$. В качестве $g_2(x, y)$ можно взять x.

4) Имеет место симметрия ответов. Действительно, пусть в задаче "3-разбиение" ответ положительный. Это означает, что множество $\{1,2,3,\ldots,3n\}$ разбито на n непересекающихся подмножеств $A_1, A_2, ..., A_{3n}$ (по 3 элемента в каждом). Каждому подмножеству A_i поставим в соответствие тройку цепочек вершин первой группы графа G, которой, в свою очередь, соответствует одна вершина второй группы. Непосредственно за каждым рабочим модулем, соответствующим таким вершинам (всего их n штук), располагаем модули контроля. Последний модуль контроля располагаем за последним рабочим модулем. Из определения A_j следует, что $\sum_{i \in A_j} a_i = B$, а значит, каждый из первых n модулей контроля контролирует B+1 вершину. Общее число вершин в графе G равно

$$nB + C_{3n}^3 + x = nB + C_{3n}^3 + B + n + 1 - C_{3n}^3 = (B+1)(n+1),$$

значит, последний модуль контроля также контролирует n+1 рабочий модуль. Получили расстановку модулей контроля, которая генерирует разбиение R графа G на равные части.

Предположим теперь что существует расстановка n+1 модулей контроля, каждый из которых контролирует B+1 рабочих модулей. Очевидно, что модуль контроля не может располагаться непосредственно после рабочего модуля, соответствующего вершине первой группы графа G (иначе он контролировал бы $y \leq a_i < B+1$ рабочих модулей). Также очевидно, что в конце цепочки рабочих модулей третей группы должен располагаться модуль контроля μ, а значит, после других вершин третей группы модулей контроля нет (иначе модуль μ контролировал бы $y \leq x < B+1$ рабочих модулей). Таким образом, n модулей контроля размещены среди C_{3n}^3 вершин второй группы, и каждый из них контролирует по B+1 вершине. Следовательно, соответствующие тройки и дадут положительный ответ в задаче "3-разбиение". Таким образом, доказано, что аналог задачи 1 в форме

распознавания свойств является *NP*-полной в сильном смысле задачей, а задача 1 – *NP*-трудной в сильном смысле.

4.5.6. Применение метода “ветвей и границ”

В этом разделе мы продолжим рассмотрение задачи 1, сформулированной в разд. 4.5.4.1. Поскольку в предыдущем разделе мы показали, что эта задача является *NP*- трудной в сильном смысле, целесообразно применить для ее решения метод “ветвей и границ”.

Пусть n рабочих модулей и связи между ними образуют произвольный граф G и пусть имеется k модулей контроля. Пусть r рабочих модулей не имеют последователей в графе G. Тогда после каждого из этих r рабочих модулей необходимо поставить модуль контроля. Следовательно, если $k < r$, то задача не имеет решения, а если $k = r$, то решение единственно. В дальнейшем будем предполагать что $k > r$.

Будем расставлять модули контроля последовательно. На первом шаге располагаем модуль контроля в произвольном месте и удаляем из графа все рабочие модули, которые он контролирует. На втором шаге располагаем следующий модуль контроля после одного из оставшихся рабочих модулей и вновь удаляем все контролируемые им модули. Повторяем эту процедуру до тех пор, пока либо не будут удалены все рабочие модули, либо число оставшихся модулей контроля не станет равным числу оставшихся рабочих модулей, не имеющих последователей в графе G. Очевидно, что в первом случае расстановка не оптимальна. Во втором случае оставшиеся модули контроля следует расположить за оставшимися рабочими модулями, не имеющими последователей.

Таким способом может быть получена любая расстановка модулей контроля. Пусть a_i – число рабочих модулей, удаленных из графа G на i-м шаге. Нам надо минимизировать величину $S = \sum_{i=1}^{k} a_i^2$. Для каждой из расстановок будем вычислять значение S. Расстановка с минимальным значением S и будет оптимальной. Число расстановок модулей контроля, получаемых описанным выше алгоритмом, вообще говоря, экспоненциально. Покажем, как уменьшить перебор с помо-

щью метода “ветвей и границ”. В дальнейшем для простоты изложения будем считать что $r=1$.

Выберем сначала первый модуль контроля так, чтобы величина $\left|a_1-\frac{n}{k}\right|$ была минимальной (т.е. чтобы величина a_1 как можно меньше отличалась от среднего из оставшихся чисел $a_1,a_2,...,a_k$), затем второй модуль контроля выбираем так, чтобы минимальной была величина $\left|a_2-\frac{n-a_1}{k-1}\right|$ (т.е. чтобы величина a_2 как можно меньше отличалась от среднего из оставшихся чисел $a_2,a_3,...,a_k$). На *i*-м шаге выбираем *i*-й модуль контроля так, что бы величина $\left|a_i-\frac{n-\sum_{j=1}^{i-1}a_j}{k-i+1}\right|$ была минимальной ($i=1,\ 2,\ 3,\ \ldots,\ k\text{-}1$). Для данной расстановки вычислим величину *S*. Будем перебирать другие расстановки с целью минимизации величины *S*. Очевидно, что на (*k*-1)-м шаге расстановку с меньшей величиной S получить нельзя (при постоянной сумме чисел a_{k-1} и a_k сумма их вкадратов минимальна, когда минимально отклонение каждого из них от среднего). Вернемся к (*k*-2)-му шагу и располагаем модуль контроля в другом месте. Пусть a'_{k-2} – новое число рабочих модулей, контролируемых указанным модулем контроля после изменения места расположения. После этого остается $x=n-\sum_{i=1}^{k-3}a_i-a'_{k-2}$ не контролируемых рабочих модулей. При любом выборе последних двух модулей контроля имеем $a'^2_{k-1}+a'^2_k\ge\frac{x^2}{2}$.

Поэтому если

$$a_1^2+...+a_{k-3}^2+a'^2_{k-2}+\frac{x^2}{2}\ge S, \qquad (16)$$

то мы не сможем улучшить величину *S* при такой расстановке первых $k-2$ модулей контроля. Следовательно, этот вариант расстановки можно исключить из рассмотрения. Более того, не трудно доказать,

что если $a'_{k-2} > \frac{x + a'_{k-2}}{3}$ (или $2a'_{k-2} - x > 0$), то для каждого a''_{k-2}, такого, что $a''_{k-2} > a'_{k-2}$, имеем

$$a''^2_{k-2} + \frac{x'^2}{2} > a'^2_{k-2} + \frac{x^2}{2}, \text{ где } x' = n - \sum_{i=1}^{k-3} a_i - a''_{k-2}. \qquad (17)$$

Действительно, обозначим $\partial = a''_{k-2} - a'_{k-2} > 0$. Тогда

$$a''^2_{k-2} + \frac{x'^2}{2} = (a'_{k-2} + \partial)^2 + \frac{(x-\partial)^2}{2} =$$

$$a'^2_{k-2} + \frac{x^2}{2} + \frac{3}{2}\partial^2 + \partial\left(2a'_{k-2} - x\right) > a'^2_{k-2} + \frac{x^2}{2}.$$

Из этих преобразований также следует, что при $a'_{k-2} < \frac{x + a'_{k-2}}{3}$ неравенство (17) верно при всех $a''_{k-2} < a'_{k-2}$.

Следовательно, если при некотором a'_{k-2} выполнено неравенство (16), то при $a'_{k-2} > \frac{x + a'_{k-2}}{3}$ можно исключить из рассмотрения также все варианты расстановки, для которых $a''_{k-2} > a'_{k-2}$, а при $a'_{k-2} < \frac{x + a'_{k-2}}{3}$ – все варианты расстановки, для которых $a''_{k-2} < a'_{k-2}$ (так как из неравенства (17) следует, что при замене a'_{k-2} на такое a''_{k-2} неравенство (16) для новой расстановки остается справедливым).

Если же при новом положении (k-2)-го модуля контроля неравенство (16) не выполнено, т. е. $a_1^2 + ... + a_{k-3}^2 + a'^2_{k-2} + \frac{x^2}{2} < S$, то надо выбрать по описанному выше алгоритму последние два модуля контроля и вычислить величину S. Если величина S уменьшилась, то все дальнейшие сравнения выполняем с этим новым значением.

После перебора всех вариантов расстановки (k-2)-го модуля контроля (при фиксированном расположении первых k-3 модулей контроля) перейдем к выбору нового расположения для (k-3)-го модуля контроля. Пусть в некотором новом расположении он контролирует a'_{k-3} рабочих модулей. Тогда после этого остается

$y = n - \sum_{i=1}^{k-4} a_i - a'_{k-3}$ не контролируемых рабочих модулей. Аналогично анализу для (k-2)-го модуля контроля заключаем, что если

$$a_1^2 + ... + a_{k-4}^2 + a'^2_{k-3} + \frac{y^2}{3} \geq S, \qquad (18)$$

то указанный вариант расстановки (k-3)-го модуля контроля (вместе со всеми соответствующими ему вариантами расстановки последних трех модулей контроля) исключаем из рассмотрения. Более того, аналогично неравенству (18) получаем, что если $a'_{k-3} > \frac{y + a'_{k-3}}{4}$ (или $3a'_{k-3} - y > 0$), то для каждого a''_{k-3}, такого, что $a''_{k-3} > a'_{k-3}$, выполняется неравенство

$$a''^2_{k-3} + \frac{y'^2}{2} > a'^2_{k-3} + \frac{y^2}{2}, \text{ где } \quad y' = n - \sum_{i=1}^{k-4} a_i - a''_{k-3}.$$

Следовательно, если для некоторого a'_{k-3} выполнено неравенство (18), то в случае, когда $a'_{k-3} > \frac{y + a'_{k-3}}{4}$, можно исключить все варианты расстановки, при которых $a_{k-3} > a'_{k-3}$. Аналогично, в случае, когда $a'_{k-3} < \frac{y + a'_{k-3}}{4}$, можно исключить все варианты расстановки, при которых $a_{k-3} < a'_{k-3}$.

Если (18) не выполнено, то расставляем последние три модуля контроля по описанному выше алгоритму с возвратом. После перебора всех вариантов расстановки (k-3)-го модуля контроля переходим к выбору новой расстановки для (k-4)-го модуля контроля и т.д.

Конечно, сложность описанного алгоритма в худшем случае экспоненциальна. Но если в начальном приближении нам удастся получить хорошую оценку величины S, то практически весь перебор будет исключен. Например, если n кратно k ($n = k\,b$) и на первом шаге удалось расположить k модулей контроля так, что $a_i = b, \quad i = 1,...,k$, то мы сразу же получаем оптимальную расстановку.

4.5.7. Альтернативная стратегия при обнаружении ошибки

В предыдущих разделах мы рассматривали задачу оптимальной расстановки модулей контроля в предположении, что при обнаружении ошибки модулем контроля рестарт системы происходит немедленно. В этом разделе мы рассмотрим пример, в котором проиллюстрируем, что такая стратегия не всегда является оптимальной.

Рассмотрим систему из 9 рабочих модулей $A_1, A_2, ..., A_9$ с зависимостями по данным (A_1, A_2), (A_3, A_4), (A_5, A_6), (A_2, A_7), (A_4, A_7), (A_2, A_8), (A_6, A_8), (A_4, A_9), (A_6, A_9). Также имеются три модуля контроля C_1, C_2, C_3. Тогда единственная допустимая расстановка состоит в том, чтобы расположить модули контроля C_1, C_2, C_3 после рабочих модулей A_7, A_8, A_9. Из симметрии следует, что для этого графа от порядка выполнения модулей контроля ничего не зависит, и все простые стратегии эквивалентны. Рассмотрим, например следующий порядок.

1) Выполняем модули $A_1, A_2, A_3, A_4, A_7, C_1$, при ошибке повторяем. Потом выполняем A_5, A_6, A_8, C_2, при ошибке повторяем. И, наконец, A_9 и C_3, при ошибке повторяем. При этом математическое ожидание времени работы системы равно

$$M_1 = (1-9a)\cdot 9 + 5a(9+5) + 3a(9+3) + a(9+1) = 9 + 35a\,.$$

Рассмотрим альтернативную стратегию, не являющуюся простой.

2) Выполняем все модули до конца, не обращая внимания на ошибки. Если все модули контроля показывают ошибки, повторяем все сначала (впрочем, для такого случая нужны ошибки одновременно как минимум в двух рабочих модулях, а вероятность этого пренебрежимо мала). Если ошибки показывают модули C_1 и C_2 (но не C_3), повторяем выполнение рабочих модулей A_3, A_4, A_7, A_8 (т.е. тех, которые не контролирует C_3). Этот случай реализуется при ошибке в модулях A_3 и A_4. Аналогично, если ошибки показывают модули C_1 и C_3 повторяем выполнение A_1, A_2, A_7, A_9, а если модули C_2 и C_3, то повторяем A_5, A_6, A_8, A_9. Если же ошибку показал только модуль C_1, надо повторить только рабочий модуль A_7 (ошибка определенно в нем).

Аналогично, если об ошибке сигнализирует C_2, надо повторить A_8, а если C_3 – повторяем A_9.

При такой стратегии математическое ожидание работы всей системы равно

$$M_2 = (1-9a)\cdot 9 + 3\cdot 2a(9+4) + 3\cdot a(9+1) = 9 + 27a < M_1.$$

Как видим, в этом примере более выгодно при обнаружении ошибки не приступать к перезапуску системы немедленно.

Во всех предыдущих разделах мы предполагали, что ошибка при выполнении рабочих модулей возможна, но маловероятна. В следующем разделе мы рассмотрим случай, когда вероятность возникновения ошибки может быть существенной.

4.6. Расстановка модулей контроля для случая произвольной вероятности ошибки

В этом разделе мы рассмотрим случай, когда не выполнено предположение 3 разд. 4.3, т.е. вероятность возникновения ошибки при выполнении каждого рабочего модуля может принимать произвольные значения $a \in (0,1)$.

Мы начнем с рассмотрения случая, когда выполнены предположения 1 и 2 раздела 4.3, т.е. n рабочих модулей образуют цепочку, и количество рабочих модулей в этой цепочке большое. Более того, будем предполагать, что имеется только один модуль-буфер, который надо расположить вначале цепочки (из условия допустимости расстановки) и k модулей контроля. Рассмотрим задачу оптимального расположения модулей контроля в этом случае. Как и ранее, произвольную расстановку модулей контроля будем обозначать $A(k,1,n)$, а оптимальную – $A_{опт}(k,1,n)$.

Получим формулы для нахождения математического ожидания времени выполнения всех рабочих модулей (с учетом повторных выполнений при возникновении ошибки) $MA(k,1,n)$ при заданной расстановке $A(k,1,n)$.

Предположим сначала, что модуль контроля расположен непосредственно после первого рабочего модуля M_1. Определим матема-

тическое ожидание M количества выполнений модуля M_1 с учетом повторений (до его выполнения без ошибки). Если модуль M_1 выполнится без ошибки, то число его выполнений равно 1, а если с ошибкой, то следует повторить выполнение этого модуля. В последнем случае мы окажемся в исходной ситуации, и поэтому среднее число выполнений модуля M_1 равно $1+M$. Таким образом,

$$M=(1-\alpha)\cdot 1+\alpha\cdot(1+M),\quad M=\frac{1}{1-\alpha}. \tag{19}$$

Пусть теперь модуль контроля C_1 расположен непосредственно после рабочего модуля M_x. Тогда вероятность того, что все модули $M_1, M_2,..., M_x$ отработают без ошибки, равна $(1-\alpha)^x$. Аналогично (19) определяется математическое ожидание M количества выполнений модулей $M_1, M_2,..., M_x$ с учетом повторений:

$$M=(1-\alpha)^x\cdot x+(1-(1-\alpha)^x)\cdot(x+M),\quad M=\frac{x}{(1-\alpha)^x}. \tag{20}$$

И наконец, пусть задана некоторая расстановка $A(k,1,n)$. Пусть k модулей контроля разбивают n рабочих модулей на группы с числом модулей $x_1, x_2,..., x_k$ $(\sum_{i=1}^{k} x_i = n)$ соответственно. Тогда вероятность обнаружения ошибки первым модулем контроля равна $1-(1-\alpha)^{x_1}$, вторым – $(1-\alpha)^{x_1}\cdot(1-(1-\alpha)^{x_2})$, j-м модулем контроля $(j=3,...,r)$ – $(1-\alpha)^{\sum_{i=1}^{j-1} x_i}\cdot(1-(1-\alpha)^{x_j})$. Вероятность того, что все рассматриваемые рабочие модули выполнятся без ошибки, равна $\sum_{i=1}^{k} x_i$. При обнаружении ошибки j-м модулем контроля нужно повторить выполнение $\sum_{i=1}^{j} x_i$ рабочих модулей. Следовательно,

$$M=(1-(1-\alpha)^{x_1})(M+x_1)+(1-\alpha)^{x_1}(1-(1-\alpha)^{x_2})\cdot(M+x_1+x_2)+...$$
$$+(1-\alpha)^{\sum_{i=1}^{k-1} x_i}\cdot(1-(1-\alpha)^{x_k})\cdot(M+\sum_{i=1}^{k} x_i)+(1-\alpha)^{\sum_{i=1}^{k} x_i}\cdot(\sum_{i=1}^{k} x_i)=$$

$$= M\,(1-(1-\alpha)^{x_1} + (1-\alpha)^{x_1} - (1-\alpha)^{x_1+x_2} + \ldots - (1-\alpha)^{\sum_{i=1}^{k} x_i}) +$$

$$+ x_1(1-(1-\alpha)^{x_1} + (1-\alpha)^{x_1} - \ldots - (1-\alpha)^{\sum_{i=1}^{k} x_i} + (1-\alpha)^{\sum_{i=1}^{k} x_i}) +$$

$$+ x_2((1-\alpha)^{x_1} - (1-\alpha)^{x_1+x_2} + (1-\alpha)^{x_1+x_2} - \ldots) + \ldots$$

$$+ x_k((1-\alpha)^{\sum_{i=1}^{k-1} x_i} - (1-\alpha)^{\sum_{i=1}^{k} x_i} + (1-\alpha)^{\sum_{i=1}^{k} x_i}) =$$

$$= M\,(1-(1-\alpha)^{\sum_{i=1}^{k} x_i}) + x_1 + x_2(1-\alpha)^{x_1} + \ldots + x_k(1-\alpha)^{\sum_{i=1}^{k-1} x_i}, \text{ или}$$

$$MA(k,1,n) = \frac{x_1}{(1-\alpha)^{\sum_{i=1}^{k} x_i}} + \frac{x_2}{(1-\alpha)^{\sum_{i=2}^{k} x_i}} + .. + \frac{x_k}{(1-\alpha)^{x_k}}. \qquad (21)$$

В следующем разделе мы разработаем алгоритм для построения расстановки, при которой это математическое ожидание минимально.

4.6.1. Оптимальное расположение модулей контроля для случая одной цепочки

В этом случае надо расставить k модулей контроля так, чтобы минимизировать величину $MA(k,1,n)$, задаваемую (21), т.е. надо найти $\min\limits_{x_i, i=1,\ldots,k} MA(k,1,n)$ при условии $\sum\limits_{i=1}^{k} x_i = n$. Для функции Лагранжа $L = MA(k,1,n) + \lambda \cdot \left(\sum\limits_{i=1}^{k} x_i - n \right)$ имеем систему уравнений:

$$\begin{cases} \dfrac{\partial L}{\partial x_1} = \dfrac{1 - x_1 \ln(1-\alpha)}{(1-\alpha)^{\sum x_i}} + \lambda = 0, \\ \dfrac{\partial L}{\partial x_2} = \dfrac{-x_1 \ln(1-\alpha)}{(1-\alpha)^{\sum x_i}} + \dfrac{1 - x_2 \ln(1-\alpha)}{(1-\alpha)^{\sum x_i - x_1}} + \lambda = 0, \\ \dfrac{\partial L}{\partial x_3} = \dfrac{-x_1 \ln(1-\alpha)}{(1-\alpha)^{\sum x_i}} + \dfrac{-x_2 \ln(1-\alpha)}{(1-\alpha)^{\sum x_i - x_1}} + \dfrac{1 - x_3 \ln(1-\alpha)}{(1-\alpha)^{\sum x_i - x_1 - x_2}} + \lambda = 0, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \dfrac{\partial L}{\partial x_k} = \dfrac{-x_1 \ln(1-\alpha)}{(1-\alpha)^{\sum x_i}} + \ldots + \dfrac{-x_{k-1} \ln(1-\alpha)}{(1-\alpha)^{x_{k-1} + x_k}} + \dfrac{1 - x_k \ln(1-\alpha)}{(1-\alpha)^{x_k}} + \lambda = 0, \\ \dfrac{\partial L}{\partial \lambda} = \sum_{i=1}^{k} x_i - n = 0. \end{cases}$$

Вычтем из каждого уравнения полученной системы (со второго до предпоследнего) предыдущее. Добавляя к полученным уравнениям первое и последнее уравнения исходной системы, получим новую систему, эквивалентную исходной:

$$\begin{cases} \dfrac{1 - x_1 \ln(1-\alpha)}{(1-\alpha)^{\sum x_i}} + \lambda = 0, \\ 1 - x_2 \ln(1-\alpha) = \dfrac{1}{(1-\alpha)^{x_1}}, \\ 1 - x_3 \ln(1-\alpha) = \dfrac{1}{(1-\alpha)^{x_2}}, \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ 1 - x_k \ln(1-\alpha) = \dfrac{1}{(1-\alpha)^{x_{k-1}}}, \\ \sum x_i - S = 0. \end{cases}$$

Из первого уравнения последней системы следует, что

$$-\lambda = \frac{1 - x_1 \ln(1-\alpha)}{(1-\alpha)^{\sum x_i}}. \qquad (22)$$

Введем функцию $f(x) = \dfrac{1 - \dfrac{1}{(1-\alpha)^x}}{\ln(1-\alpha)}$. Из последней системы следует, что $x_2 = f(x_1)$, $x_3 = f(x_2)$, …, $x_k = f(x_{k-1})$.

Подставляя эти значения в последнее уравнение системы, получим:

$x_1 + f(x_1) + f^{[2]}(x_1) + ... + f^{[k-1]}(x_1) - S = 0$, где $f^{[i]}(x) = \underbrace{f(f(...f(x)...))}_{i-раз}$.

Определим функцию

$$\varphi_k(x) = x + f(x) + f^{[2]}(x) + ... + f^{[k-1]}(x). \quad (23)$$

Рассмотрим уравнение $\varphi_k(x) = n$. Если x_1 – его решение, то значения $x_2, x_3, ..., x_k$ определяются по формуле $x_i = f(x_{i-1}), i = 2,...,k$. Заметим, что $f(x) > 0$ при $x > 0$, поскольку $0 < \alpha < 1$. Кроме того, $f'(x) = \frac{1}{(1-\alpha)^x} > 0$. Следовательно, $f(x)$ (а значит, и $\varphi(x)$) строго возрастает на интервале $(0;+\infty)$. Отметим, что $f(0) = 0$, и, следовательно, $f^{[i]}(0) = 0$ при всех $i = 1,...,k-1$, $\varphi_k(0) = 0$. Кроме того, $\lim\limits_{x\to\infty} f(x) = +\infty$, $\lim\limits_{x\to\infty} \varphi_k(x) = +\infty$. Отсюда следует, что уравнение $\varphi_k(x) = n$ имеет ровно один корень на интервале $(0;+\infty)$. Из (23) следует, что $\varphi_k(n) > n$, а следовательно, этот корень принадлежит интервалу $(0;n)$. Его можно эффективно найти, например, методом деления отрезка пополам.

Таким образом, при m=1 можно определить оптимальное расположение модулей контроля, которому соответствуют значения переменных $x_1, x_2, ..., x_k$, минимизирующие функцию $MA(k,1,n)$, задаваемую (21). Полученное оптимальное значение для математического ожидания будем, как обычно, обозначать $M_{опт}(k,1,n)$.

Заметим, что формально мы можем вычислить величину $M_{опт}(k,1,n)$ не только для натурального n, но и для любого действительного числа. Обозначим $g_k(S) = M_{опт}(k,1,S)$ для любого действительного $S > 0$. Для дальнейшего нам понадобится производная $g_k'(S)$. Рассчитаем ее.

Для этого докажем следующее свойство множителя Лагранжа. Пусть определяется минимум некоторой функции $f(x_1, x_2, ..., x_k)$ при условии $x_1 + x_2 + ... + x_k = S$ методом множителей Лагранжа. Рассмот-

рим полученный минимум как функцию от S (обозначим ее $g(S)$). Тогда $g'(S) = -\lambda$, где λ – значение множителя Лагранжа, найденного из системы, $L = f(x_1, x_2, ..., x_k) + \lambda \cdot (x_1 + x_2 + ... + x_k - S)$.

Для функции L имеем систему уравнений:

$$\begin{cases} \dfrac{\partial L}{\partial x_1} = \dfrac{\partial f}{\partial x_1} + \lambda = 0, \\ \dfrac{\partial L}{\partial x_2} = \dfrac{\partial f}{\partial x_2} + \lambda = 0, \\ ..... \\ \dfrac{\partial L}{\partial x_k} = \dfrac{\partial f}{\partial x_k} + \lambda = 0, \end{cases}$$

откуда $-\lambda = \dfrac{\partial f}{\partial x_1} = \dfrac{\partial f}{\partial x_2} = ... = \dfrac{\partial f}{\partial x_k}$. Следовательно,

$g'(S) = \sum\limits_{i=1}^{k} \dfrac{\partial f}{\partial x_i} \dfrac{\partial x_i}{\partial S} = -\lambda \sum\limits_{i=1}^{k} \dfrac{\partial x_i}{\partial S} = -\lambda \dfrac{\partial \left(\sum\limits_{i=1}^{k} x_i \right)}{\partial S} = -\lambda$. Свойство доказано.

Из (22) с учетом доказанного свойства множителей Лагранжа получаем, что

$$g'_k(S) = -\lambda = \frac{1 - x_1 \ln(1-\alpha)}{(1-\alpha)^{\sum x_i}} = \frac{1 - \varphi^{-1}{}_k(S) \ln(1-\alpha)}{(1-\alpha)^S} . \qquad (24)$$

Заметим, что поскольку $\varphi(S)$ строго возрастающая функция, то $\varphi^{-1}(S)$ также строго возрастает, а так как $\alpha \in (0,1)$, то $g'_k(S)$ также строго возрастает. Следовательно, $g''_k(S) > 0$ и $g_k(S)$ строго выпуклая функция при $S > 0$.

Так как $g'_k(0) = 0$, то

$$M_{onm}(k,1,n) = g_k(n) = \int_0^n g'_k(S) ds = \int_0^n \frac{1 - \varphi^{-1}{}_k(S) \ln(1-\alpha)}{(1-\alpha)^S} ds . \qquad (25)$$

Таким образом, имеем следующее. Пусть дано k модулей контроля и n рабочих модулей. Тогда нужно решить уравнение $\varphi_k(x) = n$, где

$\varphi_k(x)$ определяется из (23) при $f(x)=\dfrac{1-\dfrac{1}{(1-\alpha)^x}}{\ln(1-\alpha)}$. Пусть x_1 – корень этого уравнения. Вычислим $x_2=f(x_1)$, $x_3=f(x_2)$, …, $x_k=f(x_{k-1})$. Тогда оптимальная расстановка модулей контроля делит n рабочих модулей на части $x_1,x_2,...,x_k$. Математическое ожидание времени выполнения при такой расстановке задается формулой (25).

Пример 3. Пусть иммется n=500 рабочих модулей и k=5 модулей контроля. Вероятность обнаружения ошибки при выполнении каждого из рабочих модулей α=0.01. При этом предположение 3 разд. 4.3 $\left(a<<\dfrac{1}{n}\right)$ не выполнено и теория разд. 4.3. не применима.

Решая уравнение $\varphi_5(x)=500$, получим x_1=44.5577. Далее, $x_2=f(x_1)$ = 100.7643, $x_3=f(x_2)$=176.3106, $x_4=f(x_3)$=289.4118, $x_5=f(x_4)$=500. Следовательно, модули контроля надо располагать после 45-го, 101-го, 176-го, 289-го и 500-го рабочего модуля.

Математическое ожидание времени выполнения рабочих модулей (с учетов возможных повторений при ошибках) при этом будет равно $M_{опт}(5,1,500)$ = 21100.

4.6.2. Случай равных параллельных цепочек

В этом разделе мы рассмотрим случай, когда n рабочих модулей разбиты на l одинаковых цепочек по s рабочих модулей в каждой (n=ls). Будем предполагать, что имеется l модулей-буферов, которые (из условия допустимости расстановки) надо расположить в начале каждой из цепочек. Нужно оптимально распределить k модулей контроля по этим цепочкам.

Очевидно, что в таком случае модули контроля надо распределять по цепочкам равномерно. В разд. 4.4.1 мы доказали этот результат для случая малой вероятности ошибки (теорема 8). Этот результат сразу же последовал из леммы о “выпуклости” $M_{опт}(k,m,n)$ как функции от k (лемма 4). Следовательно, нам достаточно обобщить

эту лемму для случая произвольной вероятности ошибки, или, другими словами, достаточно доказать следующую теорему.

Теорема 10. Для любых натуральных k, h, n $(k > h)$ выполняется неравенство $M_{опт}(k-h,1,n) + M_{опт}(k+h,1,n) > 2M_{опт}(k,1,n)$, где $M_{опт}(k,1,n)$ задается формулой (25).

Доказательство теоремы следует из следующих свойств функции $\varphi_k(x)$, где $\varphi_k(x)$ определяется из (23) при $f(x) = \dfrac{1 - \dfrac{1}{(1-\alpha)^x}}{\ln(1-\alpha)}$.

Свойство1.

$$\varphi_k'(x) = 1 + \frac{1}{(1-\alpha)^{\varphi_1(x)}} + \frac{1}{(1-\alpha)^{\varphi_2(x)}} + \ldots + \frac{1}{(1-\alpha)^{\varphi_{k-1}(x)}}.$$

Свойство 2. Пусть $a = \varphi_{k+1}^{-1}(S), b = \varphi_k^{-1}(S)$. Тогда $f(a) < b$.

Свойство 3. $(\varphi_k^{-1}(S))' > (\varphi_{k+1}^{-1}(S))'$, для произвольного S.

Свойство 4 ("выпуклость" функции $\varphi_k^{-1}(x)$ по k).

Для всех $k \in N, S \in (0;+\infty)$ выполняется неравенство $\varphi_{k-1}^{-1}(S) + \varphi_{k+1}^{-1}(S) > 2\varphi_k^{-1}(S)$.

Свойство 5. Пусть $k_1, k_2 \in N$, причем $k_2 - 1 \geq k_1 + 1$. Тогда для всех $S \in (0;+\infty)$ выполняется неравенство $\varphi_{k_2}^{-1}(S) - \varphi_{k_2-1}^{-1}(S) > \varphi_{k_1+1}^{-1}(S) - \varphi_{k_1}^{-1}(S)$.

Отсюда следует, (см. лемму 3 и теорему 8) что в рассматриваемом случае k модулей контроля надо распределять по l цепочкам равномерно. А именно, если $k = l \cdot \bar{b} + \bar{r}, \bar{r} < l$, то можно взять $k_1 = k_2 = \ldots = k_{\bar{r}} = \bar{b} + 1$, $k_{\bar{r}+1} = \ldots = k_m = \bar{b}$.

Математическое ожидания времени выполнения всех модулей в этом случае задается выражением

$$\bar{r}M_{опт}(\bar{b}+1,1,s) + (l-\bar{r})M_{опт}(\bar{b},1,s),$$

где $M'_{опт}(\bar{b}+1,w,s)$ и $M'_{опт}(\bar{b},w,s)$ задаются формулой (25).

Заключение

В первой главе были рассмотрены различные варианты задачи о поиске допустимого расписания в системе реального времени, получены необходимые и достаточные условия существования решения и предложены эффективные алгоритмы для его нахождения.

В разд. 1.2 была рассмотрена задача поиска решений в многопроцессорной системе реального времени, в которой разрешены прерывания и переключения работ, переключения работ не требуют дополнительных затрат, связи между процессорами образуют полный граф, а ограничения по памяти отсутствуют. Для этого случая приведен известный точный полиномиальный алгоритм, который, однако, бывает затруднительно применять на практике в системах большой размерности из-за достаточно высокой степени полинома вычислительной сложности ($O\left(m^3n^3\right)$, m – число процессоров, n – число работ). Поэтому в данной главе были предложены два более быстрых эвристических алгоритма ($O\left(mn\right)$ и $O\left(n^2\log n\right)$). При помощи значительного количества (более 20000) численных экспериментов при различных значениях параметров задачи исследованы границы корректности применения предложенных эвристических алгоритмов, получена сравнительная статистическая информация относительно некорректной работы алгоритмов (второй алгоритм медленнее, чем первый, зато обеспечивает в среднем в несколько раз меньший процент некорректной работы – 3% против 20%), даны рекомендации по их практическому применению.

Во разд. 1.3 условия задачи из первой главы усложнились введением временных затрат процессоров на прерывания и возобновления работ. Было рассмотрено три различных модели образования таких затрат, при условии что во всех этих случаях затраты на прерывания и возобновления малы относительно характерной длины директивных интервалов работ. Предложенные в разд. 1.2 эвристические алгоритмы были адаптированы для этого случая и было определено количество прерываний, которое генерирует в построенном допустимом расписании каждый из этих алгоритмов (соответственно, $2(n\text{-}1)m$ и $(n\text{-}1)$). Несмотря на то, что для решения задачи из предыдущей главы второй

алгоритм был предпочтительнее в смысле корректности его работы, в этом варианте задачи из-за того, что он генерирует большее количество прерываний, чем первый, начиная с некой относительной величины прерывания первый алгоритм становится предпочтительнее. Поэтому для каждой из трех моделей образования затрат на переключения было исследовано, при каких значениях параметров, задающих отношение величины прерываний к характеристической длине директивных интервалов работ, первый алгоритм становится предпочтительнее второго в смысле более высокой вероятности корректности его работы.

В разд. 1.4 рассмотрен вариант задачи о поиске допустимого расписания в многопроцессорной системе с ограничениями по памяти процессоров и конечной скоростью ввода-вывода данных, в котором процессоры, помимо скорости выполнения работ, характеризуются скоростью загрузки данных. Рассмотрен случай одинаковых директивных интервалов для всех работ. Для однопроцессорного случая построен точный алгоритм, решающий задачу за полиномиальное время $O\left(n^2 \log n\right)$ и теоретически обоснована его корректность. Далее рассмотрен многопроцессорный случай этой задачи, доказана ее NP-трудность как в случае, когда переключения работ с процессора на процессор разрешены, так и в случае, когда они не допускаются. Поэтому для ее практического решения был разработан эвристический алгоритм вычислительной сложности $O(n^2 m \log n)$, котрый, как показали проведенные численные эксперименты, эффективно решает задачу, в случае, когда максимальные длительности выполнения работ и загрузки данных существенно меньше, чем длина общего директивного интервала всех работ. Представлено теоретическое объяснение этого полученного эмпирически свойства алгоритма, даны рекомендации по его практическому применению. Для выявления случаев некорректной работы эвристического алгоритма для рассмотренной в данной главе задачи был также построен точный алгоритм и дано теоретическое обоснование его корректности.

В разд. 1.5 рассмотрена задача поиска допустимого расписания в многопроцессорной системе реального времени с ограничениями по объему памяти процессоров в случае неполного графа связей между

процессорами. Для некоторого упрощения условий задачи считалось, что все процессоры работают дискретными синхронизованными по времени тактами. Для решения задачи построена многопродуктовая потоковая сеть специального вида и доказано, что решение исходной задачи существует тогда и только тогда, когда в построенной сети существует многопродуктовый поток, обладающий рядом указанных свойств. Показано, что построение расписания сводится к нахождению такого потока, которое, в свою очередь, сводится к решению задачи булевого линейного программирования. Указаны наиболее эффективные псевдополиномиальные алгоритмы решения такой задачи.

В разд. 1.6 работы рассмотрена задача синтеза, т. е. задача построения многопроцессорной системы, в которой при заданном множестве работ допустимое расписание с прерываниями заведомо существует. Построена система неравенств, задающая допустимую с точки зрения существования допустимого расписания область производительности процессоров в случае, когда ограничения на объем памяти процессоров отсутствуют. Были указаны методы решения возникающих в связи с этим различных оптимизационных задач (например, минимизация суммы производительностей процессоров), которые в данном случае сводятся к решению задачи целочисленного линейного программирования. Рассмотрена задача синтеза системы реального времени в случае ограничений по объему памяти процессоров (считается, что загрузка и выгрузка данных работ из памяти процессоров осуществляется за пренебрежимо малое время). Была получена система неравенств, связывающая искомые производительности процессоров и их объемы памяти с параметрами работ. Доказано, что выполнение этой системы является необходимым и достаточным условием для существования в рассмотренной системе допустимого расписания с прерываниями, то есть решения поставленной задачи синтеза. Были указаны методы решения возникающих здесь различных оптимизационных задач, которые в данном случае сводятся к решению задачи целочисленного квадратичного программирования с булевскими переменными.

Наиболее перспективным представляется развитие предложенных в данной главе методов и подходов по следующим направлениям:

- построение эффективных эвристических алгоритмов, решающих задачу поиска допустимого расписания в постановке разд. 1.5 (многопроцессорная система с ограничениями по памяти и неполным графом связей между процессорами) существенно быстрее, чем указанный там псевдополиномиальный точный алгоритм, неприменимый в практических задачах большой размерности;

- подробное рассмотрение различных оптимизационных задач синтеза систем с ограничениями по памяти процессоров, построение более эффективных алгоритмов для их решения, чем указанные в данной работе алгоритмы общего характера.

Во второй главе были рассмотрены различные варианты задачи составления расписания без прерываний, разработаны алгоритмы решения задачи с идентичными и произвольными процессорами, определены их временная сложность и погрешность получаемого расписания. Основные результаты главы заключаются в следующем.

1. Разработан подход к решению задачи составления расписания с идентичными процессорами, заключающий в укрупнении первоначальных требований системы путем декомпозиции задачи, независимом (последовательном или параллельном) решении подзадач и агрегировании решений полученных решений в требования первоначальной системы. Для исследования и апробации предложенного метода был разработан пакет прикладных программ и проведено большое количество экспериментов. Результаты экспериментов демонстрируют высокую эффективность предложенного метода и его практическую важность.

2. Среди разработанных агрегирующих методик выявлены наиболее перспективные, которые за допустимое для большинства практических задач время находят точное или близкое к точному решение.

3. Для задачи составления расписания без прерываний с произвольными процессорами разработаны эвристические и точный псевдополиномиальный алгоритмы. Для них также созданы комплексы вычислительных программ и получено большое число экспериментальных данных. Результаты проведенных экспериментов позволили сравнить разработанные в данной работе алгоритмы с методом имитации отжига. Полученные данные позволяют заключить, что пред-

ложенные алгоритмы эффективнее последнего как по быстродействию, так и по точности получаемого решения.

4. Для разработанных алгоритмов ПАПГ, ВА, ПРОП, СДРП аналитически были получены верхние оценки погрешности получаемых решений.

5. Доказано, что задача составления расписания без прерываний на фиксированном числе произвольных процессоров с пустым отношением предшествования является псевдополиномиально разрешимой. Таким образом, многие практические задачи рассматриваемого класса, директивные сроки в которых равномерно ограничены константой, могут быть решены алгоритмом ПАПШ за полиномиальное время.

6. Предложен ряд параллельных вычислительных алгоритмов с высокой эффективностью распараллеливания. Результаты, полученные на параллельном вычислительном кластере, показывают большую практическую значимость разработанных алгоритмов. Полученные в результате вычислительных экспериментов значения эффективности распараллеливания близки к единице, а функция изоэффективности алгоритмов является линейной, что означает их хорошую масштабируемость.

Наиболее перспективным представляется развитие предложенных в данной главе методов и подходов по следующим направлениям:

- разработка новых методик агрегирования для задачи составления расписаний в общей постановке (с произвольными процессорами);

- исследование новых стратегий распараллеливания псевдополиномиальных алгоритмов и применение их в других параллельных архитектурах;

- определение зависимости погрешности и алгоритмической сложности агрегирующих алгоритмов от параметров исходной системы и декомпозиции.

В третьей главе получены следующие результаты.

1. Рассмотрена задача составления допустимого расписания с прерываниями в многопроцессорной системе в случае, когда заданы директивные интервалы, а длительности выполнения работ линейно

зависят от количества выделенного им дополнительного ресурса. Решение задачи сведено к задаче линейного программирования и к задаче нахождения потока минимальной стоимости. Решена задача оптимальной коррекции директивных интервалов в случае, когда допустимого расписания не существует.

2. Решена задача построения допустимого расписания с прерываниями для случая, когда имеется несколько различных по производительности процессоров, работы допускают прерывания, а требования на выполнение работ поступают циклически с заданными периодами. Разработан полиномиальный алгоритм, основанный на сведении исходной задачи к поиску максимального потока в сети специального вида.

3. Разработан алгоритм построения допустимых расписаний выполнения работ в системе реального времени при наличии неопределенных факторов. Задача сведена к антагонистической игре специального вида.

В четвертой главе получены следующие результаты.

В задаче оптимизации структуры подсистемы контроля в вычислительных системах реального времени разработаны алгоритмы для построения оптимальных расстановок модулей контроля и модулей-буферов в случаях, когда граф зависимости по данным является цепочкой или состоит из нескольких параллельных цепочек. Также разработан полиномиальный алгоритм расстановки модулей контроля для случая, когда граф зависимости по данным является ориентированным деревом, модули-буферы располагаются после каждого рабочего модуля, а ошибка при выполнении рабочего модуля возможна, но маловероятна.

В задаче оптимальной расстановки модулей контроля и модулей-буферов остается широкое поле деятельности для дальнейшего исследования. Во-первых, это обобщение всех результатов, полученных для малой вероятности ошибки, на случай, когда вероятность возникновения ошибки произвольна. Далее интересно исследовать случай, когда вероятность возникновения ошибки различна для различных заданий, меняется во времени, либо вообще неизвестна.

Второе направление – выделение других типов графов (кроме цепочки, параллельных цепочек, и дерева), при котором задача не является NP-трудной. При этом желательно снять ограничение, что после каждого рабочего модуля располагается модуль-буфер.

Третье направление – рассмотрение многопроцессорной системы. При этом можно рассматривать процессоры разной производительности, а рабочие модули – разной сложности. В этом случае нужно не только оптимально расставить модули контроля и модули-буфера, но после этого синхронизировать работу различных процессоров, чтобы они оптимальным образом дополняли друг друга. Кроме того, во многих случаях имеет смысл отказаться от предположения, что рестарт происходит сразу же после обнаружения ошибки. В этом случае возникает еще и задача построения оптимальной стратегии рестартов.

Библиография

1. *Танаев В.С., Гордон В.С., Шафранский Я.М.* Теория расписаний. Одностадийные системы. – М.: Наука, 1984.
2. *Головкин Б.А.* Расчет характеристик и планирование параллельных вычислительных процессов. – М.: Радио и Связь, 1983.
3. *Барский А.Б.* Параллельные процессы в вычислительных системах. Планирование и организация. – М.: Радио и связь, 1990.
4. *C. Martel,* Preemptive scheduling with release times, deadlines and due times // Journal of the ACM. 1982. V. 29, №3. P. 812–829.
5. *K. Ramamritham and J. A. Stankovic*, Scheduling Algorithms and Operating Systems Support for Real–Time Systems, Proceedings of the IEEE, 82(1): 55–67, Jan 1994.
6. *A. Burns*, Scheduling Hard Real-Time Systems: A Review, Software Engineering Journal, 6(3): 116–128, May 1991.
7. *J.A. Stankovic, et. al.*, Implications of Classical Scheduling Results for Real-Time Systems, IEEE Computer Society Press, 1995.
8. *H. Tokuda, T. Nakajima and P. Rao*, Real-Time Mach: Toward a Predictable Real-Time System, Proceedings of USENOX Mach Workshop, Oct 1990.
9. *Lynx Programmer's Reference Manual*, Version 2.4, Lynx Real-Time Systems, San Jose, CA, 1996.
10. *C.L. Liu and J.W. Layland*, Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment, Journal of the ACM, 20(1): 46–61, Jan 1973.
11. *A. Federgruen, H. Groenevelt*. "Preemptive Scheduling of Uniform Machines by Ordinary Network Flow Technique". Management Science Vol. 32, No. 3, March 1986.
12. *T. Gonzales, S. Sanhi*. "Preemptive Scheduling of Uniform Processor Systems". Journal of the Association for Computing Machinery, Vol. 25, No. 1, January 1978.
13. *Т. Кормен, Ч. Лейзерсон, Р. Ривест*. «Алгоритмы: построение и анализ» М. МЦНМО, 1999.

14. *Э.Г. Коффман*. Введение в детерминированную теорию расписаний. В кн.: Теория расписаний и вычислительные машины / Под ред. Коффмана Э.Г. – М. Наука, 1984, с. 9–64.
15. *Мину М*. Математическое программирование. Теория и алгоритмы. М.: Наука, 1990.
16. *P.M. Vaidya.* Speeding up linear programming using fast matrix multiplication. In “Proceedings of the 30th Annual Symposium on Foundations of Computer Science”, pages 332–337, 1989.
17. *Логинова И.В., Сушков Б.Г.* Динамическое распределение памяти в системах реального времени при имеющемся расписании центрального процессора // В сборнике «Теория и реализация систем реального времени», ВЦ АН СССР, стр. 49–69, 1984.
18. *Фуругян М.Г.* Некоторые алгоритмы анализа детерминированных систем реального времени. М. ВЦ РАН – 1989.
19. *E.C. Horvath, S. Lam, R. Sethi.* A level algorithm for preemptive scheduling. J. ACM 24 (Jan 1977), 32–43.
20. *Малков У.Х.* О реализации методов Вульфа и ветвей и границ для решения частично-целочисленных задач квадратичного программирования // Тезисы доклада на научной конференции "Математическое программирование и приложения", Екатеринбург: ИММ Уральского научного центра РАН, 1996.
21. *Киселев В.Д., Карелин Д.В.* Метод ветвей и границ для решения задач целочисленного квадратичного программирования с булевыми переменными // Известия Тульского Государственного Университета, 1995, Том 1, выпуск 3, Информатика.
22. *A.K.Mok.* Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment, Ph.D Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1983.
23. *N.Audsley, A.Burns, M.Richardson and A.Wellings* Hard Real-Time Scheduling: The Deadline Monotonic Approach, IEEE Workshop on Real-Time Operating Systems, 1992.
24. *M.L.Dertouzos* Control Robotics: The Procedural Control of Physical Processes, Information Processing 74, North-Holland Publishing Company, 1974.

25. *Hessel F., Coste P., Nicolescu G., LeMarrec P., Zergainoh N., Jerraya A. A.* Communication Synthesis of Multilanguage Specification // Research Report, TIMA Laboratory, ISRN TIMA-RR-00/06-1-FR, ISSN 1292–8062.
26. *Baghdady A., Zergainoh N-E., Cesario W., Roudier T., Jerraya A. Design* Space Exploration for Hardware/Software Codesign of Multiprocessor Architectures // Research Report, TIMA Laboratory, ISRN TIMA-RR-00/02-4-FR, ISSN 1292–8062.
27. Сверхбольше интегральные схемы и современная обработка сигналов // Под ред. Гуна С., Уайтхауза Х., Кайлата Т., – М.: Радио и связь, 1989.
28. *Майоров С.А., Новиков Г.И.* Принципы организации цифровых машин – Л.: Машиностроение, 1974.
29. *Valderrama C. et al*. Hardware and Software Co-design : Principles and Practice KLUWER. Chap. COSMOS : A Transformational Codesign Tool for Multiprocessor Architectures, 1997, p. 307-357.
30. *Ernst R., Henkel J., Benner Th., Trawny M*. The COSYMA Environment for Hardware/Software Cosynthesis, Journal of Microprocessors and Microsystems, Butterworth-Heinemann, 1995.
31. *Ben-Ismail T., O'Brien K., Jerraya A.* PAR-TIF: Interactive System-level Partitioning, VLSI Design Vol. 3 №3-4, pp. 333-345, 1995.
32. *Костенко В.А., Трекин А.Г.* Генетические алгоритмы решения смешанных задач целочисленной и комбинаторной оптимизации при синтезе архитектур ВС // Искусственный интеллект (Донецк), 2000, №2, С.90–96.
33. *Костенко В.А., Смелянский Р.Л., Трекин А.Г.* Генетические алгоритмы: синтез структур вычислительных систем // Тезисы докладов Всероссийской научной конференции "Фундаментальные и прикладные аспекты разработки больших распределенных программных комплексов” М.: Изд-во МГУ, 1998, С.35–41.
34. *Костенко В.А., Смелянский Р.Л., Трекин А.Г.* Синтез структур вычислительных систем реального времени с использованием генетических алгоритмов// Программирование, 2000, №5.

35. *Трекин А.Г.* Структурный синтез вычислительных систем с помощью генетических алгоритмов // диссертация на соискание ученой степени кадидата физ.-мат. наук, - М., 2002.
36. *Гуз Д.С.* Быстрые алгоритмы составления расписаний в многопроцессорных системах //Современные проблемы фундаментальных и прикладных наук. Часть VII. Прикладная математика и экономика: Труды XLV научной конференции. /Моск. физ. – техн. ин-т. – М. – Долгопрудный, 2002. – С. 56–57.
37. *Гуз Д.С.*, *Фуругян М.Г.* Разработка и сравнительный анализ точных и приближенных алгоритмов составления расписаний с прерываниями в многопроцессорных системах // Моделирование обработки информации и процессов управления: Сб.ст./Моск.физ.-тех. ин-т. – М., 2002. – С. 296–306.
38. *Гончар Д.Р., Гуз Д.С.*, *Красовский Д.В., Фуругян М.Г.* Эффективные алгоритмы планирования вычислений в многопроцессорных системах. // Проблемы управления безопасностью сложных систем: Труды 10-й международной конференции. Книга 2. /ИПУ РАН. – М., 2002 – С. 207–211.
39. *Гуз Д.С.*, *Красовский Д.В., Фуругян М.Г.* Некоторые алгоритмы составления расписаний в многопроцессорных системах //Проблемы управления безопасностью сложных систем: Труды 11-й международной конференции. /ИПУ РАН. – М., 2003 – С. 12–14.
40. *Гуз Д.С.*, *Фуругян М.Г.* Сведение задачи о поиске допустимого расписания в многопроцессорной системе реального времени к задаче о многопродуктовом потоке в сети // Моделирование и обработка информации: Сб.ст./Моск.физ.-тех. ин-т. – М., 2003. – С. 87–95.
41. *Гуз Д.С.*, *Фуругян М.Г.* Возможность применения алгоритма поиска многопродуктового потока для составления расписаний в многопроцессорных системах жёсткого реального времени //Современные проблемы фундаментальных и прикладных наук. Часть VII. Прикладная математика и экономика: Труды XLVI научной конференции. /Моск. физ. – техн. ин-т. – М. – Долгопрудный, 2003. – С. 65–66.

42. *Гуз Д.С.*, *Красовский Д.В.*, *Фуругян М.Г.* Эффективные алгоритмы планирования вычислений в многопроцессорных системах реального времени: Препринт / ВЦ РАН. – М., 2004. –65 с.
43. *Гуз Д.С.*, *Фуругян М.Г.* Составление расписаний выполнения заданий и загрузки памяти для однопроцессорных систем жесткого реального времени // Моделирование процессов управления: Сб.ст./Моск.физ.-тех. ин-т. – М., 2004. – С. 162–169.
44. *Гуз Д.С.*, *Фуругян М.Г.* Алгоритмы составления допустимых расписаний для системы жесткого реального времени с ограничениями по объему памяти //Проблемы управления безопасностью сложных систем: Труды 12-й международной конференции. /ИПУ РАН. – М., 2004 – С. 96–98.
45. *Guz D., Krasovskiy D., Furugian M.* Effective scheduling algorithms for multiprocessor real-time systems //Proceedings of IV Moscow International Conference on Operations Research. /ВЦ РАН. – М., 2004 – С. 100–103.
46. *Гуз Д.С.*, *Фуругян М.Г.* Разработка эффективных алгоритмов планирования вычислений в системах жесткого реального времени с учетом ограничений по памяти //Современные проблемы фундаментальных и прикладных наук. Часть VII. Прикладная математика и экономика: Труды XLVII научной конференции. /Моск. физ. – техн. ин-т. – М. – Долгопрудный, 2004. – С. 96–98.
47. *Гуз Д.С.*, *Фуругян М.Г.* Планирование вычислений в многопроцессорных АСУ реального времени с ограничениями на память процессоров // Автоматика и телемеханика. – 2005. – №2 – С. 138–147.
48. *Гуз Д.С.*, *Фуругян М.Г.* Эвристический алгоритм составления расписаний для многопроцессорных систем с ограничениями по объему памяти // Процессы и методы обработки информации: Сб.ст./Моск.физ.-тех. ин-т. – М., 2005. – С. 4–10.
49. *Логинова И.В.* Алгоритмы динамического распределения памяти в системах реального времени // автореферат диссертации на соискание ученой степени кандидата физико-математических наук – М., 1985.

50. *Ульман Дж.* Полиномиально полные задачи составления расписаний // Operating Systems Review, 1973.
51. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи // М.: Мир, 1982.
52. *Brucker P.* Scheduling algorithms, 2nd edition // Springer, Heidelberg, 1998.
53. *Bondy J. L., Freeman D. N.* Putting supervisory routines into hardware // North-Holland Publ. Co., 1977.
54. *Сигал И.Х., Иванова А.П.* Введение в прикладное дискретное программирование // ФИЗМАТЛИТ. 2002.
55. *Хачатуров В.*Р., Веселовский В.Е., Злотов А.В. и др. Комбинаторные методы и алгоритмы решения задач дискретной оптимизации большой размерности // М.: Наука, 2000.
56. *Сигал И.Х.* Параметризация и исследование некоторых задач дискретного программирования большой размерности // М.: Известия академии наук. Теория и системы управления, 2001, №2, С. 60–69.
57. *Сигал И.Х.* Параметризация ε-приближенных алгоритмов решения некоторых классов задач дискретной оптимизации большой размерности // М.: Известия академии наук. Теория и системы управления, 2002, №6, С. 63–72.
58. *Сигал И.Х.* Оценки параметров алгоритмов ветвей и границ для задач дискретной оптимизации большой размерности // М.: Известия академии наук. Теория и системы управления, 2005, №4, С. 96–101.
59. *Baker K. R.* Introduction to Sequencing and Scheduling // John Wiley and Sons, Inc., 1974.
60. *Conway R., Maxwell W., Miller L.* Theory of Scheduling // Addison Wesley Publishing Company, 1967.
61. *Giffler B., Thompson G.* Algorithms for solving production scheduling problems // Operations Research, 8(4):487–503, 1960.
62. *Растригин Л.А.* Статистические методы поиска // М.: Наука, 1968.

63. *Гончаров Е.Н., Кочетов Ю.А.* Вероятностный поиск с запретами для дискретных задач безусловной оптимизации // Дискрет. анализ и исслед. операций Сер. 2, 2002. Т. 9, №2. С. 13–30.
64. *Кочетов Ю, Младенович Н., Хансен П.* Локальный поиск с чередующимися окрестностями // Дискрет. анализ и исслед. операций Сер. 2, 2003. Т. 10, № 1, С. 11–44.
65. *Кочетов Ю.А., Столяр А.А.* Использование чередующихся окрестностей для приближенного решения задачи календарного планирования с ограниченными ресурсами // Дискрет. анализ и исслед. операций Сер. 2, 2003. Т. 10, № 2, С. 29–56
66. *Tripathy A.* School timetabling – a case in large binary integer linear programming // Management Science, 30(2):1473–1480, 1984.
67. *Ciriani T., Leachman R., editors.* Optimizing in Industry: Mathematical Programming and Modeling Techniques in Practice // John Wiley and Sons, Ltd., 1993.
68. *Алексеев О.Г.* Комплексное применение методов дискретной оптимизации // Наука, 1986.
69. *Held M., Karp R.* A dynamic programming approach to sequencing problem // SIAM, 1962.
70. *Land A.H., and Doig A.G.* An automatic method of solving discrete programming problems // Econometrica. v28 (1960), pp 497–520.
71. *Литтл Д.Ж., Мурти К., Суини Д., Кэрел К.* Алгоритм для решения задачи коммивояжера // Экономика и математические методы, Т. 1, вып. 1, С. 90–107, 1965.
72. *Корбут А.А., Финкельштейн Ю.Ю.* Дискретное программирование // М. Наука. Гл. ред. физ.-мат. лит. 1969.
73. *Panwalkar S., Iskander W.* A survey of scheduling rules // Operations Research. 25(1):45–61, 1977.
74. *Штовба С. Д.* Муравьиные алгоритмы // ExponentaPro. Математика в приложениях, № 4(4), 2003.
75. *Laarhoven P., Aarts E., Lenstra J.* Job Shop Scheduling by Simulated Annealing // Operations Research, 40(1):113–125, 1992.
76. *Shen C., Pao Y., Yip P.* Scheduling multiple job problems with guided evolutionary simulated annealing approach // Proceedings of

the First IEEE Conference on Evolutionary Computations, pages 702–706, 1994.

77. *Glover F., Laguna M.* Chapter 3: Tabu search // In Colin R. Reeves, editor, Modern Heuristics Techniques for Combinatorial Problems, Blackwell Scientific Publications, 1993.
78. *Taillard E.* Benchmarks for basic scheduling problems // European Journal of Operations Research, 64:278–285, 1993.
79. *Ross P., Hallam J.* Lecture Notes on Connectionist Computing // Department of Artificial Intelligence, University of Edinburgh, 1993.
80. *Hulle M.* A goal programming network for mixed integer linear programming: A case study for the job-shop scheduling problem // International Journal of Neural Systems, 2(3):201–209, 1991.
81. *Меламед И.И.* Нейронные сети и комбинаторная оптимизация // Автоматика и телемеханика, 11, С. 3–40, 1994.
82. *Holland J.N.* Adaptation in Natural and Artificial Systems // Ann Arbor, Michigan: Univ. of Michigan Press, 1975.
83. *Michalewicz Z.* Genetic Algorithms + Data Structures = Evolution Programs // Third, Revised and Extended Edition, Springer, 1999.
84. *Goldberg D.E.* Genetic Algorithms in Search Optimization & Machine Learning // Addison Wesley, Reading, 1989.
85. *Mataric M., Cliff D.* Challenges in Evolving Controllers for Physical Robots // Robotics and autonomous systems, 19(1), p. 67–83, 1996.
86. *Periaux J. and Winter G. editors.* Genetic Algorithms in Engineering and Computer Science // John Wiley & Sons Ltd., 1995.
87. *Corne D., Fang H.-L., Mellish C.* Solving the Modular Exam Scheduling Problem with Genetic Algorithms // DAI Research Paper №622.
88. *Bierwirth C., Kopfer H., Mattfeld D.C., Rixen I.* Genetic Algorithm based Scheduling in a Dynamic Manufacturing Environment // Proceedings of the IEEE Conf. on Evolutionary Computation, Perth, IEEE Press, 1995, p.439–443.
89. *Fang H.-L., Ross P., Corne D.* A Promising genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling, and Open-Shop Scheduling Problems // Proceedings of the Fifth International Con-

ference on Genetic Algorithms, S. Forrest (ed.), San Mateo: Morgan Kaufmann, 1993, p.375–382.

90. *Daaldr J., Eklund P.W., Ohmori K.* High-Level Synthesis Optimization with Genetic Algorithms // Proceedings of the 4th Pacific Rim International Conference on Artificial Intelligence, Cairns (Australia), 26–30 August 1996, p.276–287.
91. *Посыпкин М.А., Сигал И.Х., Гамильянова Н.Н.* Алгоритмы параллельных вычислений для решения некоторых классов задач дискретной оптимизации. М.: ВЦ РАН, 2005.
92. *Rao V. Nageshwara and Kumar V.* Parallel depth-first search, part I: Implementation // International Journal of Parallel Programming, 1987, 16 (6). P. 479–499.
93. *Rao V. Nageshwara and Kumar V.* Parallel depth-first search, part II: Analysis // International Journal of Parallel Programming, 1987, 16(6). P. 501–519.
94. *Kumar V.*, *Grama A.*, *Rao V. Nageshwara.* Scalable Load balancing Techniques for Parallel Computers // Journal of Parallel and Distributed Computing, 1994, 22(1). P. 60–79.
95. *Gotlieb A.* The NYU ultracomputer – designing a MIMD, shared memory parallel processor // IEEE Transactions on Computers, February, 1983. P. 175– 189.
96. *Тимошевская Н. Е.* Параллельные методы обхода дерева// Математическое моделирование. 2004, 16(4). С.105–114.
97. *Kouichi Kimura*, *Ichiyoshi Nobuyuki.* Probabilistic analysis of the efficiency of the dynamic load distribution // Sixth Distributed Memory Computing Conference Proceedings, 1991. P. 145–152.
98. *Shu W.*, *Kale L. V.* A dynamic scheduling strategy for the chare-kernel system // Supercomputing, 1989. P. 389–398.
99. *Ranade A.* Optimal speedup for backtrack search on butterfly network // ACM Symposium on Parallel Algorithms and Architectures, 1991. P. 40–48.
100. *Raghavan R.* Probabilistic Construction of Deterministic Algorithms: Approximating Packing Integer Programs // Journal of Computer and System Sciences 37, 130–143, 1988.

101. *Сигал И.Х.* Задача коммивояжера большой размерности // ВЦ АН СССР, 1986.

102. *Michael J. Quinn.* Designing Efficient Algorithms for Parallel Computers // McGraw-Hill, 1987.

103. *Гончар Д.Р., Гуз Д.С., Красовский Д.В., Фуругян М.Г.* Эффективные алгоритмы планирования вычислений в многопроцессорных системах. // Проблемы управления безопасностью сложных систем: Труды 10-й международной конференции. Книга 2. /ИПУ РАН. – М., 2002 – С. 207–211.

104. *Красовский Д.В., Фуругян М.Г.* Комплексное применение методов дискретной оптимизации при решении задач минимаксной теории расписаний // МФТИ – М., 2003. «Моделирование и обработка информации» – С. 96–103.

105. *Гуз Д.С., Красовский Д.В., Фуругян М.Г.* Некоторые алгоритмы анализа многопроцессорных систем реального времени / ВЦ РАН. – М., 2005. – 42 с.

106. *Красовский Д. В.* Решение задачи составления оптимального расписания без прерываний на произвольных процессорах с использованием вероятностного алгоритма // МФТИ – М., 2005. Сборник трудов 48 научной конференции МФТИ, С. 76–78.

107. *Гуз Д.С., Красовский Д.В., Фуругян М.Г.* Эффективные алгоритмы планирования вычислений в многопроцессорных системах реального времени // Методы и средства обработки информации. Труды второй всероссийской научной конференции. – Москва, издательский отдел факультета ВМиК МГУ. 2005 г., –С. 540–545.

108. *Красовский Д.В.*, *Фуругян М.Г.* Агрегирование в задаче составления оптимального расписания для многопроцессорных АСУ // Автоматика и телемеханика. – 2006. – №12 – С. 205–212.

109. *Красовский Д.В.*, *Фуругян М.Г.* Псевдополиномиальные алгоритмы упорядочения работ без прерываний по произвольным процессорам // Вестник Московского университета, сер. 15, Вычислительная математика и кибернетика, 2006, № 4, – С. 25–29

110. *Красовский Д. В.* Алгоритм параллельного планирования работы многопроцессорных систем большой размерности // Совре-

менные проблемы фундаментальных и прикладных наук – общая и прикладная физика: Сборник трудов 49-й научной конференции МФТИ, Т. II / МФТИ – М.: 2006. – С. 79–80.

111. *Давыдов Э.Г.* Исследование операций. М.: Высшая школа, 1990.
112. *Филлипс Д., Гарсиа-Диас А.* Методы анализа сетей. М.: Мир, 1984.
113. *Фуругян М.Г., Косоруков Е.О.* Некоторые алгоритмы распределения ресурсов в многопроцессорных системах. // Вестник МГУ, сер. 15, 2009, № 4. С. 34–37.
114. *Косоруков Е.*О., *Фуругян М.Г.* Некоторые алгоритмы распределения ресурсов в системах с нефиксированными длительностями работ. М.: ВЦ РАН, 2009.
115. *Косоруков Е.О., Фуругян М.Г.* Алгоритмы распределения ресурсов в многопроцессорных системах с нефиксированными параметрами // В сб. «Некоторые алгоритмы планирования вычислений и организации контроля в системах реального времени». М.: ВЦ РАН, 2011. С. 40–51.
116. *Пападимитриу Х., Стайглиц К.* Комбинаторная оптимизация. Алгоритмы и сложность. М.: Мир, 1985. С. 316 – 351.
117. *Фуругян М.Г.* Приближенное решение одного класса бесконечных антагонистических игр с полунепрерывной платежной функцией. Вестн. МГУ. Сер. Вычисл. матем. и киберн. 1980. N 2. С. 66 – 69.
118. *Раевич С.*К. Применение метода "ветвей и границ" для решения одной задачи распределения ресурсов. "Мат. методы решения экономических задач". М.: ЦЭМИ АН СССР, 1969. N 1. С. 114 – 126.
119. *Мищенко А.В., Сушков Б.Г.* Минимизация времени выполнения работ, представленных сетевой моделью, при нефиксированных параметрах сети. М.: ВЦ АН СССР, 1980.
120. *Chandy K.* A survey of analytic models of roll-back and recovery strategies. // Computer 8,5, 1975, pp. 40–47.
121. *Young J. W.* A first-order approximation to the optimum checkpoint interval. // Comm. ACM 17,9, 1974, pp. 530–531.

122. *Gelembe E.* On the Optimum Checkpoint Interval // J. ACM, vol. 26, pp. 259–270 1979.
123. *Duda A.* The Effects of Checkpointing on Program Execution Time. // Information Processing Letters, vol. 16, no. 5, pp. 221–229.
124. *Grassi V., Donatiello L., Tucci S.,* On the Optimal Checkpointing of Critical Tasks and Transaction-Oriented Systems. // IEEE Trans. Software Eng., vol. 18, no. 1, pp. 72–77, Jan. 1992.
125. *Leung C., Choo Q.* On the Execution of Large Batch Programs in Unreliable Computing Systems, // IEEE Trans. Software Eng., vol. 10, no. 4, pp. 444–450, July 1984.
126. *Coffman E., Gilbert E.* Optimal Strategies for Scheduling Checkpoints and Preventive Maintenance, // IEEE Trans. Reliability, vol. 39, no. 1, pp. 9–18, Apr. 1990.
127. *Benczur A, Kramlin A.* An example for an adaptive control method providing database integrity. // In Proceedings of the Fourth International Symposium on Modeling and Performance Evaluation of Computer Systems, pp. 249–262, 1979.
128. *Tantawi A., Ruschitzka M.* Performance Analysis of Checkpointing Strategies. // ACM Trans. Computer Systems, vol. 2, no. 2, pp. 123–144, 1984.
129. *Toueg S., Babaoglu O.* On the Optimum Checkpoint Selection Problem // SIAM J. Computing, vol. 13, no. 3, pp. 630–649, Aug. 1984.
130. *Bruno J.L., Coffman E.G.* Optimal Fault-Tolerant Computing on Multiprocessor Systems // Acta Informatica, vol. 34, pp. 881–904, 1997.
131. *Ecuyer, Malenfant.* Computing optimal checkpointing strategies for rollback and recovery systems // *IEEE Trans. Computers*, C-37 (4), pp. 491–496.
132. *Tatsuya Ozaki, Tadashi Dohi, Hiroyuki Okamura, Naoto Kaio.* Min-Max Checkpoint Placement under Incomplete Failure Information. // International Conference on DSN, 2004.
133. *Луганская М.И., Сушков Б.Г.* Контроль данных в системах реального времени. // Математические методы управления обработкой информации. М.: МФТИ, 1986. С. 18 – 24.

134. *Белый Д.В., Сушков Б.Г.* Модель организации рестартов в системах реального времени. М.: ВЦ РАН, 1996.
135. *Гречук Б.В., Фуругян М.Г.* Алгоритмы организации рестартов в системах реального времени. М.: ВЦ РАН, 2004.
136. *Гречук Б.В., Фуругян М.Г.* Алгоритмы организации рестартов в системах реального времени с произвольным графом связей. М.: ВЦ РАН, 2004.