УЧРЕЖДЕНИЕ РОССИЙСКОЙ АКАДЕМИИ НАУК
ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР
ИМ. А.А. ДОРОДНИЦЫНА

НЕКОТОРЫЕ АЛГОРИТМЫ ПЛАНИРОВАНИЯ ВЫЧИСЛЕНИЙ В МНОГОПРОЦЕССОРНЫХ СИСТЕМАХ РЕАЛЬНОГО ВРЕМЕНИ

ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР ИМ. А.А. ДОРОДНИЦЫНА
РОССИЙСКОЙ АКАДЕМИИ НАУК
МОСКВА 2010

УДК 519.86

Ответственный редактор
канд. физ.-матем. наук Л.Л. Вышинский

В сборнике приводится краткое описание системы автоматизации проектирования вычислительных систем реального времени. Разработан приближенный алгоритм составления оптимального по быстродействию расписания на произвольных процессорах множества работ, часть из которых допускает прерывания и переключения с одного процессора на другой, а часть – не допускает. Рассматривается задача составления допустимого расписания с прерываниями в многопроцессорной системе в случае, когда длительности выполнения работ линейно зависят от количества выделенного им дополнительного ресурса. Исследуется задача оптимизации структуры подсистемы контроля в вычислительных системах реального времени. Предлагается один алгоритм синтеза многопроцессорной системы. Исследованы два представления графов системой прямоугольников. Определен класс графов, допускающих представление соприкасающимися прямоугольниками. Исследован класс планарных триангуляций.

Ключевые слова: вычислительная система реального времени, многопроцессорная система, оптимальное расписание, прерываемые и непрерываемые работы, ресурс, контроль, синтез, представление графов прямоугольниками.

Рецензенты: *В.А. Серебряков,*
Д.А. Кононов

Научное издание

ПРОБЛЕМЫ АВТОМАТИЗАЦИИ ПРОЕКТИРОВАНИЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ

Д.Р. Гончар, С.Н.Мирошник, М.Г. Фуругян

Приводится краткое описание системы автоматизации проектирования вычислительных систем реального времени. Рассматриваются основные блоки и язык системы. Формулируются основные проблемы проектирования систем реального времени.

1. Введение

Вычислительные системы реального времени предназначены для контроля и управления процессами в автоматизированных производствах, транспортных системах, системах производства энергии, в частности, атомных электростанциях, в нефте- и газодобывающих производствах, в химической промышленности и многих других областях человеческой деятельности.

В секторе проектирования систем реального времени ВЦ РАН разработана инструментальная система автоматизации программирования вычислительных систем реального времени (САПР ВСРВ) для IBM PC. Система предназначена для автоматизации проектирования и генерации систем реального времени, осуществляющих обработку циклически поступающей информации в темпе поступления при жестких временных ограничениях. Эта система, во-первых, позволяет в короткие сроки в автоматическом режиме проектировать из готовых прикладных модулей конкретную ВСРВ, описанную пользователем с помощью программы реального времени (РВ-программы), и, во-вторых, заранее, т.е. до обработки инфор-

мации в реальном времени, проводить оптимизацию вычислений. Для реализации такого подхода необходимо предварительно построить математическую модель ВСРВ. С помощью этой модели могут быть решены такие задачи, как разбиение программ на параллельные процессы, составление допустимого расписания выполнения процессов, синхронизация вычислений, динамическое распределение памяти. При разработке этой системы был получен ряд вспомогательных результатов, которые представляют отдельный теоретический и практический интерес. Например, были разработаны и реализованы наглядные языковые средства для описания циклической обработки информации, разработаны различные алгоритмы дискретной оптимизации и составления расписаний для ВСРВ.

Для генерации прикладной ВСРВ пользователю необходимо иметь прикладные модули, написанные на языках программирования, например Си, Фортран, Паскаль или Ассемблер, и написать на специальном языке реального времени задание на обработку информации в реальном времени – РВ-программу. В этой программе пользователь задает порядок обработки прикладными модулями входных данных и отображения результатов счета по отношению к периоду поступления кадров данных в систему. Если данный порядок обработки может быть соблюден, система обеспечит реализацию заказанной обработки. В противном случае выдается сообщение о невозможности вести указанную обработку. Предусмотрена возможность работы прикладных модулей с несколькими поколениями данных. Система автоматически обеспечивает хранение этих данных в специальных буферах нужное время. Предусмотрена также возможность быстрой реакции на поступление апериодической информации. Это может быть использовано, например, при возникновении внештатной ситуации с управляемым объектом, либо для изменения порядка работы программы реального времени оператором. Также предусмотрено выполнение прикладных модулей в фоновом режи-

ме. Вся остальная работа по генерации прикладной ВСРВ будет выполнена автоматически с помощью разработанной САПР ВСРВ. При этом будут решены следующие проблемы:

1) проблема синхронизации параллельных процессов в ВСРВ, под которыми понимаются как процессы функционирования периферийных устройств системы, являющиеся внешними по отношению к используемой вычислительной системе, так и вычислительные процессы, реализуемые выполнением РВ-программы;

2) проблема тупиков при распределении ресурсов вычислительной системы (процессоров, каналов, памяти, периферийных устройств, а также информационных ресурсов, под которыми понимаются используемые в режиме разделения вспомогательные программы и информационные массивы);

3) проблема завершения тех или иных вычислений к определенным моментам времени или к моментам определенных событий в системе, указанных пользователем в РВ-программе (соблюдение директивных сроков);

4) проблема сохранения и обновления необходимых наборов поколений данных, как поступающих в ВСРВ извне, так и являющихся результатами вычислений с помощью РВ-программы, для последующего использования в вычислениях;

5) проблема обнаружения в режиме реального времени ошибок и сбоев в работе датчиков, каналов передачи данных к ВСРВ и внутри самой вычислительной системы, а также организации соответствующих реакций на обнаруженные ошибки и сбои.

Решение этих задач обеспечивает надежность программного обеспечения ВСРВ. Реализация разработанной САПР ВСРВ является результатом применения простого принципа: максимальная подготовка программы реального времени к выполнению до режима реального времени. Разработанная система наиболее эффективна при составлении программ для детерминированных ВСРВ с циклическим поступлением инфор-

мации от контролируемого объекта. Детерминированность означает здесь следующее:

1) входная информация поступает в систему с известным периодом T, т.е. в моменты времени 0, T, $2T$, ...;

2) входная информация может быть объединена в кадры, имеющие постоянную часть (т.е. поступающую в каждый дискретный момент 0, T, $2T$, ...) и несколько переменных частей, каждая из которых поступает в систему реже, с большим периодом, кратным T. Например, одна переменная часть кадра может поступать в систему с периодом $2T$, т.е. в моменты 0, $2T$, $4T$, ..., другая - с периодом $5T$, т.е. в моменты времени 0, $5T$, $10T$, ... Количество переменных частей кадра произвольно;

3) контролируемый процесс состоит из конечного числа детерминированных участков. Для каждого такого участка заранее известно, какие данные необходимо получать от объекта и передавать к нему, определены форматы входных кадров, а также известно, какую обработку необходимо производить, т.е. задан список программных модулей осуществляющих нужные вычисления. Иными словами, для каждого участка контролируемого процесса известен режим обработки. Участки могут следовать друг за другом в произвольном порядке и сменять друг друга в моменты времени, определяемые как результатами контроля, так и выполненных вычислений;

4) для каждого программного модуля, выполняющего вычисления или некоторый обмен данными, известна оценка времени исполнения этого модуля, а также потребности в других ресурсах системы, например, требуемый объем оперативной памяти, объем дисковой памяти, потребность в других ресурсах системы;

5) в процессе выполнения каждого режима обработки может быть вычислен момент смены текущего режима и однозначно определен следующий режим обработки;

6) задан начальный (стартовый) режим обработки (например, режим ожидания первого кадра информации от контролируемого объекта).

При разработке САПР ВСРВ для IBM PC были реализованы (1) язык реального времени; (2) транслятор с этого языка, включающий блоки синтаксического и семантического анализа, построения сетевой модели вычислений и нахождения допустимого расписания выполнения вычислений, автоматической генерации объектного кода программы реального времени; (3) управляющий монитор, контролирующий вычислениями в режиме реального времени. Остановимся кратко на описании языка реального времени и основных блоков САПР ВСРВ.

2. Язык реального времени

Синтаксис языка реального времени, опуская не существенные для дальнейшего детали, может быть описан следующим образом. Основным элементарным объектом языка является модуль – обычный для многих языков программирования оператор процедуры. Все процедуры, участвующие в образовании модулей, составляются заранее и помещаются в системную библиотеку процедур вместе с необходимыми спецификациями формальных параметров. Имеется один специальный тип модуля, введенный для облегчения составления РВ-программ – это РВ-модуль. Он по описанию и смыслу ничем не отличается от обычного модуля. Но кроме описания он еще имеет тело, внутри которого указаны вызовы других, в том числе и РВ-модулей. Для РВ-модулей имеется ограничение – они не должны содержать рекурсивных вызовов.

Фактические параметры любого модуля могут быть РВ-параметрами, константами, простыми переменными, идентификаторами с индексами и т.д. РВ-параметр определяется именем, поставщиком и поколением. Поставщик однозначно указывает программный модуль, в котором вычисляется за-

прашиваемое модулем-потребителем значение параметра с данным именем. Как будет видно из дальнейшего описания языка, вызовы модулей могут повторяться в программе, а сами модули могут входить в более сложные объекты языка. Поэтому эта компонента, в свою очередь, состоит из нескольких полей. Если имя поставщика отсутствует, то считается, что этот РВ-параметр присутствует во входных данных. Компонента “поколение” определяет момент времени, в который необходимо взять требуемое значение параметра. В этом разделе может быть указано время, кратное периодичности поступления информации от указанного поставщика данных. Системой будет передано потребителю последнее имеющееся на заданный момент времени значение этого параметра.

Из модулей может быть составлен следующий по иерархии сложности объект языка – цикл реального времени, который состоит из имени РВ-цикла, периода, фазы и тела РВ-цикла. Компонента “период” задает интервал времени, через который будет повторяться выполнение модулей, указанных в теле РВ-цикла. Задание периода аналогично заданию времени для РВ-параметра. Если в программе реального времени указано несколько источников данных или в качестве базового периода берется период другого РВ-цикла, то требуется явное указание его имени перед значением периода. Кроме этого период может быть задан в абсолютных величинах времени: секундах, миллисекундах. Параметр “фаза” указывает сдвиг начала работы РВ-цикла относительно начала работы программы. Правила задания параметра “фаза”, такие же, как и для “периода”. Фаза не может быть задана больше периода, указанного для данного РВ-цикла. Если фаза явно не указана, то она равна нулю. Тело РВ-цикла - это список модулей, РВ-параметры которых могут поставляться из блоков входной информации, из модулей других РВ-циклов и из модулей данного цикла.

Поименованная совокупность РВ циклов образует безусловное, или простое, задание. Безусловное задание может быть снабжено конструкциями предварительной и заключительной части. Они служат для проведения набора специфических действий перед началом работы и соответственно после конца работы простого задания. Здесь может проходить инициализация переменных, каналов связи, переключение режима работы датчиков и т.д. Для облегчения программирования на языке реального времени, в новое простое задание допускается вставить несколько других простых заданий. В этом случае не придется дублировать исходный текст ранее определенного простого задания. Основную часть простого задания составляют РВ-циклы. Все циклы реального времени, входящие в одно простое задание, должны рассматриваться как выполняющиеся параллельно во времени. Если модули некоторого цикла требуют на свой вход данные, поставляемые другим циклом, то необходимая задержка организуется системой.

В ходе обработки эксперимента может потребоваться изменить режим обработки. Такая возможность предусматривается в языке реального времени введением условного задания, которое состоит из имени задания, вектора условий, таблицы переключений, флагов и очередей и списка простых заданий. В разделе “вектор условий” определяется список булевых переменных, от значения которых будет зависеть порядок исполнения программы реального времени. Здесь же указываются начальные значения компонент вектора. Компоненты вектора условий могут быть использованы в качестве входных и выходных параметров РВ-модулям. В разделе “таблица переключений” задаются правила переключения между простыми и условными заданиями, в зависимости от конкретных значений вектора условий. Таблица переключений представляется в виде списка значений вектора условий и имени простого или условного задания, на которое требуется переключится. Таблица переключений должна однозначно определять, на какое

задание должно происходить переключение, и содержать строку, соответствующую начальному значению вектора условий. Если для текущего значения вектора условий нет соответствующей записи в таблице переключений, то перехода на другое задание не происходит и продолжается работа программы в старом режиме.

Наконец, РВ-программа представляет собой совокупность условных заданий, все таблицы условий которой не содержат ссылок на неописанные задания.

3. Основные блоки транслятора

Основными блоками САПР ВСРВ являются блок синтаксического и семантического анализа, блок генерации сетевой модели и расписаний, блок генерации кода и управляющий монитор.

Блок синтаксического и семантического анализа осуществляет синтаксический и семантический анализ конструкций РВ-программы, выдает сообщения о наличии ошибок в РВ-программе, генерирует таблицы данных для работы последующих блоков, вычисляет размеры буферов обмена данными между программными модулями.

Блок генерации сетевой модели и расписаний строит математическую модель вычислений, выполняемых в реальном времени, в виде графа, в котором вершины соответствуют прикладным модулям пользователя, а дуги определяют частичный порядок их выполнения, определяет директивные интервалы выполнения прикладных модулей, определяет, существует ли допустимое расписание выполнения прикладных модулей, и строит его, если оно существует, определяет необходимое количество физических копий для каждого прикладного модуля, вычисляет размеры буферов для входных параметров прикладных модулей, назначает стеки прикладным модулям для работы с данными. Данный блок является одним из основных блоков системы. В последующих статьях настояще-

го сборника приводятся некоторые алгоритмы планирования вычислений в многопроцессорных САПР ВСРВ.

Блок генерации кода порождает на языке С и записывает в текущий каталог исходные тексты получившейся программы, создает ряд вспомогательных файлов для последующих определенных действий пользователя, компиляции и редактирования связей.

Управляющий монитор обеспечивает работу в реальном времени прикладной программы пользователя, сгенерированной на этапе предварительной обработки посредством САПР ВСРВ. Управляющий монитор является составной частью исполняемого EXE-модуля РВ-программы. Его функциями являются: прием, хранение и обработка поступающих извне больших порций информации (кадров данных); реакция на внешнее апериодическое сообщение, исполнение команд, вводимых с клавиатуры консоли; переключение между условными и простыми заданиями в соответствии со значениями системного вектора условий; запуск процессов согласно директивным срокам и приоритетам; обмен данными между процессами; действия по окончании работы процесса.

АЛГОРИТМЫ ПЛАНИРОВАНИЯ ВЫЧИСЛЕНИЙ В МНОГОПРОЦЕССОРНЫХ СИСТЕМАХ С НЕОДНОРОДНЫМ МНОЖЕСТВОМ РАБОТ

Д.Р. Гончар, М.Г. Фуругян

Разработан приближенный алгоритм составления оптимального по быстродействию расписания на произвольных процессорах множества работ, часть из которых допускает прерывания и переключения с одного процессора на другой, а часть – не допускает.

1. Постановка задачи

Рассматривается вычислительная система, состоящая из m процессоров, производительности которых равны s_1, s_2, …, s_m. Работая в течение времени t, процессор j выполняет работу объемом ts_j. Имеется множество работ $N = N_1 \cup N_2$, где N_1 – непрерываемые работы, N_2 – работы, допускающие прерывания и переключения с одного процессора на другой. Заданы объемы w_i и v_i работ $i \in N_1$ и $i \in N_2$ соответственно. Прерывания и переключения не связаны с временными затратами. Требуется составить оптимальное по быстродействию расписание выполнения множества работ N. Иными словами, необходимо так распределить работы по процессорам, чтобы длина временного интервала занятости наиболее загруженного процессора была минимальной.

Подобные задачи для случая, когда все работы являются непрерываемыми и не допускают переключений с одного процессора на другой, широко освещены в литературе. Отметим, например, такие методы, применяемые при их решении, как случайный и исчерпывающий поиск [1], методы математического программирования [2], метод ветвей и границ [3], му-

равьиные алгоритмы [4], поиск с запретами [5], вероятностные алгоритмы [6], генетические алгоритмы [7], метод имитации отжига [8], различные эвристические алгоритмы [9], алгоритмы агрегирования [10] и др. Задачи составления расписания прерываемых работ рассмотрены в [11–13] и др. Задачи со смешанным типом работ мало освещены в литературе. Так, например, в [14, 15] предполагается, что каждая работа строго закреплена за конкретным процессором, на множестве работ задан частичный порядок выполнения и, кроме того, только один из приборов допускает прерывания. В [16] рассмотрены случаи, когда директивные интервалы одинаковые, а также, когда директивные интервалы могут различаться, но с рядом дополнительных ограничений.

2. Разбиение процессоров на две группы

Сначала все процессоры разбиваются на две группы. Все непрерываемые работы выполняются только процессорами первой группы, а прерываемые работы будут выполняться процессорами как первой, так и второй групп. Число процессоров в группах – m_1 и m_2 – пропорционально суммарным объемам работ из N_1 и N_2 соответственно и обратно пропорционально суммарным производительностям процессоров в этих группах, т.е.

$$m_1 = \max\left\{ k : k - \text{целое}, \frac{\sum\limits_{i\in N_1} w_i}{\sum\limits_{i\in N_1} w_i + \sum\limits_{i\in N_2} v_i} \ge \frac{\sum\limits_{j=1}^{k} s_j}{\sum\limits_{j=1}^{m} s_j} \right\}, \quad m_2 = m - m_1. \quad (1)$$

Будем предполагать, что в первую группу входят процессоры $j = 1, 2, \ldots, m_1$, а во вторую – $j = m_1+1, m_1+2, \ldots, m$.

3. Распределение непрерываемых работ

Для распределения непрерываемых работ по m_1 процессорам авторами был использован приближенный мультиоценочный алгоритм с калибровкой [17], который дает неплохую оценку погрешности и является достаточно эффективным. Рекомендации по его использованию даны в [17].

Без ограничения общности можно считать, что процессоры упорядочены по не возрастанию производительностей, а работы из N_2 – по не возрастанию объемов, т.е. $s_1 \ge s_2 \ge \ldots \ge s_m$, $v_1 \ge v_2 \ge \ldots \ge v_{n_2}$. Пусть Q_j – длина интервала загруженности процессора j $(1 \le j \le m_1)$ по расписанию, построенному для множества работ N_1, и пусть

$$Q_{\max} = \max_{j=1,\ldots,m_1} Q_j, \quad \tau_{\max} = v_1 / s_m, \quad \text{для } T \ge \max(Q_{\max}, \tau_{\max})$$

$$L_j = T - Q_j \ (j = 1,\ldots,m_1), \quad n_1 = |N_1|, \quad n_2 = |N_2|.$$

4. Достаточное условие существования расписания заданной длины

Определим достаточное условие существования расписания, длина которого не превосходит величины $T \ge \max(Q_{\max}, \tau_{\max})$. Для этого сначала опишем алгоритм упаковки множества работ N_2 на (m_2+1) процессорах. Пусть $\max_{j-1,\ldots,m_1} L_j s_j = L_{j_0} s_{j_0}$ ($L_j s_j$ – максимальный объем работы, который процессор j может выполнить до момента времени T после выполнения работ из N_1). Алгоритм распределяет работы из N_2 по процессорам $j_0, m_1+1, \ldots, m$.

Алгоритм упаковки

Шаг 1. Если $\sum_{i \in N_2} v_i \le L_{j_0} s_{j_0}$, то все работы из N_2 выполнять на процессоре j_0 последовательно, начиная с момента Q_{j_0}.

В противном случае перейти на шаг 2.

Шаг 2. Пусть номер k $(0 \le k \le n_2)$ такой, что $\sum_{i=1}^{k} v_i \le L_{j_0} s_{j_0}$, а $\sum_{i=1}^{k+1} v_i > L_{j_0} s_{j_0}$.

Положить $\Theta = Q_{j_0} + (\sum_{i=1}^{k} v_i)/s_{j_0}$. Работы 1, 2, …, $k \in N_2$ выполнять последовательно без прерываний на процессоре j_0 в интервале $[Q_{j_0}, \Theta]$. Работу $k+1 \in N_2$ выполнять сначала на процессоре m_1+1 в интервале $[0, (v_{k+1} - (T-\Theta)s_{j_0})/s_{m_1+1}]$, а затем на процессоре j_0 в интервале $[\Theta, T]$. Такое переключение корректно, так как $(v_{k+1} - (T-\Theta)s_{j_0})/s_{m_1+1} \le \Theta$, что, в свою очередь, следует из соотношений

$$(v_{k+1} - (T-\Theta)s_{j_0})/s_{m_1+1} = v_{k+1}/s_{m_1+1} - (T-\Theta)s_{j_0}/s_{m_1+1} \le$$

$$\le v_1/s_m - (T-\Theta) = \tau_{\max} - T + \Theta \le T - T + \Theta = \Theta.$$

Положить $\Theta = (v_{k+1} - (T-\Theta)s_{j_0})/s_{m_1+1}$.

Шаг 3. Пусть работы $i = 1, 2, \ldots, p$ из N_2 $(k+1 \le p < n_2)$ уже распределены по процессорам $m_1+1, \ldots, l$ $(m_1 < l \le m)$, причем процессоры $m_1+1, \ldots, l$-1 загружены в интервале времени $[0, T]$, а процессор l – в интервале $[0, \Theta]$.

Если $\Theta + v_{p+1}/s_l \le T$, то работу $p+1$ выполнять без прерываний на процессоре l в интервале $[\Theta, \Theta + v_{p+1}/s_l]$; положить $\Theta = \Theta + v_{p+1}/s_l$. Если $\Theta + v_{p+1}/s_l > T$, то работу $p+1$ выполнять сначала процессором $l+1$ в интервале $[0, (v_{p+1} - (T-\Theta)s_l)/s_{l+1}] \le \Theta$, а затем процессором l в интервале $[\Theta, T]$; (такое переключение корректно, так как $(v_{p+1} - (T-\Theta)s_l)/s_{l+1} \le \Theta$, что, в свою очередь, следует из соотношений

$$(v_{p+1} - (T-\Theta)s_l)/s_{l+1} = v_{p+1}/s_{l+1} - (T-\Theta)s_l/s_{l+1} \le$$

$$\le v_1 / s_m - (T - \Theta) = \tau_{\max} - T + \Theta \le T - T + \Theta = \Theta;$$

положить $\Theta = (v_{p+1} - (T - \Theta)s_l) / s_{l+1}$.

Далее шаг 3 выполнять для работ p+2, …, n_2 из N_2.

Лемма 1. Достаточным условием существования расписания длины, не превышающей $T \ge \max(Q_{\max}, \tau_{\max})$, является выполнение неравенства

$$\sum_{i \in N_2} v_i \le L_{j_0} s_{j_0} + T \sum_{j=m_1+1}^{m} s_j. \qquad (2)$$

Доказательство леммы следует из описанного выше алгоритма упаковки.

Переписав соотношение (2) в виде

$$\sum_{i \in N_2} v_i \le (T - Q_{j_0}) s_{j_0} + T \sum_{j=m_1+1}^{m} s_j ,$$

получаем, что при

$$T \ge \frac{\sum_{i \in N_2} v_i + Q_{j_0} s_{j_0}}{\sum_{j=m_1+1}^{m} s_j + s_{j_0}}$$

существует расписание, длина которого не превосходит T. Отсюда следует, что достаточным условием существования расписания длины T является выполнение неравенства

$$T \ge T_{\max} = \max\left(\frac{\sum_{i \in N_2} v_i + Q_{\max} s_1}{\sum_{j=m_1+1}^{m} s_j + s_{m_1}},\ Q_{\max},\ \tau_{\max} \right). \qquad (3)$$

5. Необходимое условие существования расписания заданной длины

Лемма 2. Необходимым условием существования расписания длины, не превосходящей T, является выполнение неравенства

$$\sum_{i\in N_1} w_i + \sum_{i\in N_2} v_i \le T\sum_{j=1}^{m} s_j \qquad (4)$$

(при условии, что работы из N_1 уже распределены).

Доказательство леммы следует из того, что невыполнение условия (4) означает превышение величины требуемого суммарного объема работы процессоров над величиной максимально возможного объема в интервале длиной T.

Из леммы 2 следует, что не существует расписания, длина которого меньше величины

$$T_{\min} = \max\left(\frac{\sum_{i\in N_1} w_i + \sum_{i\in N_2} v_i}{\sum_{j=1}^{m} s_j},\ Q_{\max},\ \tau_{\max} \right). \qquad (5)$$

6. Модифицированный алгоритм упаковки

Алгоритм распределяет работы множества N_2 сначала по процессорам первой группы (если это возможно), а затем – по процессорам второй группы. При этом предполагается, что работы из N_1 были уже распределены по процессорам первой группы. Длина построенного расписания не превосходит T. Если такого расписания не существует, алгоритм сообщает об этом.

Шаг 1. Расположить процессоры первой группы в порядке не убывания величин $L_j s_j$, а работы из N_2 – в порядке не

убывания объемов. Будем считать, что $L_1 s_1 \le L_2 s_2 \le \ldots \le L_{m_1} s_{m_1}$; $v_1 \le v_2 \le \ldots \le v_{n_2}$.

Шаг 2. Положить $p = 1,\ j = 1$.

Шаг 3. Пусть $p,\ p+1,\ \ldots,\ n_2 \in N_2$ – работы, ранее не назначенные на процессоры. Если существует номер $k \ge p$ такой, что $\sum_{i=p}^{k} v_i \le L_j s_j$ и $\sum_{i=p}^{k+1} v_i > L_j s_j$, то назначить работы $p,\ldots,$ $k \in N_2$ на процессор j, на котором они должны выполняться последовательно в интервале $\left[Q_j, Q_j + (\sum_{i=p}^{k} v_i)/s_j \right]$. Положить $L_j = L_j - (\sum_{i=p}^{k} v_i)/s_j$; $Q_j = Q_j + (\sum_{i=p}^{k} v_i)/s_j$; $p = k + 1$. Если $p > n$, то остановиться, все работы из N_2 назначены. Если $p \le n$, перейти на шаг 4.

Шаг 4. Положить $j = j + 1$. Если $j \le m_1$, то перейти на шаг 3; если $j > m_1$ – на шаг 5.

Шаг 5. Расположить процессоры первой группы в порядке не возрастания величин $L_j s_j$. Будем считать, что $L_1 s_1 \ge L_2 s_2 \ge \ldots \ge L_{m_1} s_{m_1} > 0$ (если $L_j = 0$ при некоторых j, то соответствующие процессоры первой группы исключаем из дальнейшего рассмотрения). Положить $\Theta = 0;\quad j_1 = 1;$ $j_2 = m_1 + 1$.

Шаг 6. Если $j_1 \le m_1$ и $\Theta + (v_p - L_{j_1} s_{j_1})/s_{j_2} \le Q_{j_1}$, перейти на шаг 7;

если $j_1 \le m_1,\ \Theta + (v_p - L_{j_1} s_{j_1})/s_{j_2} > Q_{j_1}$ и $\Theta + v_p / s_{j_2} \le T$, перейти на шаг 8;

если $j_1 \le m_1,\ \Theta + (v_p - L_{j_1} s_{j_1})/s_{j_2} > Q_{j_1}$ и $\Theta + v_p / s_{j_2} > T$, перейти на шаг 9;

если $j_1 > m_1$ и $\Theta + v_p / s_{j_2} \le T$, перейти на шаг 8;

если $j_1 > m_1$ и $\Theta + v_p / s_{j_2} > T$, перейти на шаг 9;

Шаг 7. Работу p назначить на процессор j_2 в интервале $[\Theta, \Theta + (v_p - L_{j_1} s_{j_1}) / s_{j_2}]$ и на процессор j_1 в интервале $[Q_{j_1}, T]$. Положить $\Theta = \Theta + (v_p - L_{j_1} s_{j_1}) / s_{j_2}$; $p = p + 1$; $j_1 = j_1 + 1$.

Перейти на шаг 11.

Шаг 8. Работу p назначить на процессор j_2 в интервале $[\Theta, \Theta + v_p / s_{j_2}]$. Положить $\Theta = \Theta + v_p / s_{j_2}$.

Перейти на шаг 11.

Шаг 9. Работу p назначить на процессор j_2+1 в интервале $[0, (v_p - (T - \Theta) s_{j_2}) / s_{j_2+1}]$ и на процессор j_2 в интервале $[\Theta, T]$. Положить $\Theta = (v_p - (T - \Theta) s_{j_2}) / s_{j_2+1}$; $j_2 = j_2 + 1$; $p = p + 1$.

Перейти на шаг 10.

Шаг 10. Если $j_2 \le m_2$, перейти на шаг 11. Если $j_2 > m_2$, остановиться; не все работы из N_2 могут быть назначены на процессоры; расписание длины не более T не построено.

Шаг 11. Если $p > n$, остановиться; все работы из N_2 назначены на процессоры. Если $p \le n$, перейти на шаг 6.

Сделаем несколько замечаний. На шаге 3 алгоритма некоторые работы из N_2 назначаются на процессоры первой группы, на которых они выполняются без прерываний. На шаге 7 очередная работа $p \in N_2$ выполняется сначала на процессоре j_2 второй группы, а затем – на процессоре j_1 первой группы. Корректность такого переключения следует из того, что $\Theta + (v_p - L_{j_1} s_{j_1}) / s_{j_2} \le Q_{j_1}$. На шаге 9 работа p выполняется сначала на процессоре $j_2 + 1$, а затем на процессоре j_2. Такое переключение корректно, поскольку

$(v_p - (T - \Theta) s_{j_2}) / s_{j_2+1} \le \Theta$.

Это неравенство следует из соотношений:

$$(v_p - (T-\Theta)s_{j_2}) / s_{j_2+1} = v_p / s_{j_2+1} - (T-\Theta)s_{j_2} / s_{j_2+1} \le$$
$$v_1 / s_m - (T-\Theta) = \tau_{\max} - T + \Theta \le T - T + \Theta = \Theta.$$

7. Алгоритм решения исходной задачи

Поскольку решается задача на быстродействие, будем искать такое значение T^*, что расписание длины T^* существует, а расписания длины $T^* - 1$ не существует. Используя леммы 1, 2 и формулы (3), (5), этот поиск будем проводить в интервале $[T_{min}, T_{max}]$ с помощью дихотомической процедуры (деление отрезка пополам). При этом для выбранного значения T будем использовать модифицированный алгоритм упаковки.

Алгоритм решения исходной задачи

Шаг 1. По формулам (1) вычислить величины m_1, m_2.

Шаг 2. С помощью мультиоценочного алгоритма с калибровкой [17] построить расписание выполнения работ N_1 на m_1 процессорах.

Шаг 3. По формулам (3), (5) вычислить величины T_{min}, T_{max}.

Шаг 4. С помощью алгоритма деления отрезка $[T_{min}, T_{max}]$ пополам найти такое целое $\tilde{T} \in [T_{min}, T_{max}]$, что при $T = \tilde{T}$ модифицированный алгоритм упаковки строит расписание длины, не превосходящей T, а при $T = \tilde{T} - \delta$ – нет, где $0 < \delta \le 1$.

Расписание, построенное с помощью модифицированного алгоритма упаковки при $T = \tilde{T}$, – это искомое расписание выполнения работ из N_2.

Отметим, что вычислительная сложность мультиоценочного алгоритма с калибровкой (шаг 2) – $O(n_1 \log n_1)$, а модифицированного алгоритма упаковки – $O(n_2 \log n_2)$. Сложность шагов 1 и 3 – $O(n_1 + n_2)$. Поскольку

$$T_{max} \le T'_{\max} = \max\left(\frac{\sum_{i \in N_1} \tau_i + Q_{\min}}{m_2 + 1},\ Q_{\max},\ \tau_{\max} \right),$$

то число обращений к модифицированному алгоритму упаковки не более $\log (T'_{\max} - T_{\min})$. Таким образом, сложность предложенного алгоритма составляет

$$O(n_1 \log n_1 + n_2 \log \mathrm{n}_2 \cdot \log(T'_{\max} - T_{\min})).$$

8. Результаты численных экспериментов

В табл. 1–3 приведены результаты численных экспериментов. Число работ n и число процессоров m полагались равными n = 100, m = 20 в табл. 1, n = 400, m = 60 – в табл. 2 и n = 1000, m = 100 – в табл. 3. Эксперименты проводились для различных значений числа n_1 непрерываемых работ и числа n_2 прерываемых работ. Для каждого набора значений n, m, n_1, n_2 проводилось по 50 экспериментов с произвольными значениями длительностей работ, полученных с помощью программного генератора случайных чисел, позволяющего получать псевдослучайные числа с равномерным распределением на отрезке [1, 2600]. В каждом эксперименте вычислялись значения m_1 и m_2, задаваемые формулами (1), и среднее значение Δ оценки погрешности (по 50 расчетам) для каждого набора n, m, n_1, n_2. Оценка относительной погрешности алгоритма вычислялась по формуле $\Delta = \left(\frac{\tilde{T} - T^*}{T^*} \right) \times 100\ \%$, где

$$T^* = \left(\frac{\sum_{i \in N_1} w_i + \sum_{i \in N_2} v_i}{\sum_{j=1}^{m} s_j} \right).$$

(Очевидно, что T^* не превосходит длины оптимального расписания.)

Далее аналогичные расчеты проводились для всевозможных разбиений процессоров на две группы, соответствующих значениям m_1' и m_2' $(m_1' + m_2' = m)$, и для каждого такого разбиения вычислялось среднее значение оценки погрешности Δ'. В табл. 1 – 3 указаны значения m_1' и m_2' для случаев, в которых $\Delta' < \Delta$, а также значения Δ' и $\Delta < \Delta'$. Кроме того, для каждого набора *n, m,* n_1, n_2 указано значение *K*, равное проценту числа экспериментов (из 50), в которых $\Delta' < \Delta$.

Табл. 1. Результаты расчетов при $\boldsymbol{n = 100, m = 20}$.

n_1	n_2	m_1	m_2	*K*	m_1'	m_2'	Средние погрешности		
							Δ	Δ'	Δ-Δ'
1	99			0	-	-			-
10	90	2	18	12	3	17	1,9	0,3	1,7
20	80	3-4	16-17	6	4-5	15-16	1,9	0,4	1,5
30	70	5-7	15-13	6	6-8	12-14	1,4	1,0	0,4
40	60	8	12	2	9	11	1,4	1,0	0,4
50	50	10-12	8-10	12	9-11	9-11	1,9	1,5	0,3
60	40	12-13	7-8	10	11-12	8-9	3,9	2,7	1,2
70	30	14-16	4-6	26	13-15	5-7	4,6	2,7	1,9
80	20	16-17	3-4	52	15-16	4-5	6,9	3,3	3,6
90	10	18-19	1-2	64	17-18	2-3	12,2	3,2	9,0
99	1	20	0	88	19	1	20,0	5,2	14,8

Табл.2. Результаты расчетов при $n = 400$, $m = 60$.

n_1	N_2	m_1	m_2	K	m_1'	m_2'	Средние погрешности		
							Δ	Δ'	Δ-Δ'
1	399	0-1	59-60	0	-	-			
50	350	7-8	52-53	8	8-9	51-52	0,5	0,1	0,4
100	300	15	45	8	16	44	0,7	0,4	0,3
150	250	21-23	37-39	10	22-24	36-38	0,7	0,4	0,2
200	200	31-32	28-29	6	30-32	28-30	1,0	0,8	0,2
250	150	38-40	20-22	28	37-39	21-23	1,6	1,1	0,5
300	100	44-47	13-16	52	43-36	14-17	2,4	1,5	0,9
350	50	52-55	5-8	68	51-54	6-9	3,9	1,5	2,4
399	1	60	0	90	59	1	14,8	2,2	12,6

Табл.3. Результаты расчетов при $n = 1000$, $m = 100$.

n_1	N_2	m_1	m_2	K	m_1'	m_2'	Средние погрешности		
							Δ	Δ'	Δ-Δ'
1	999	0	100	0					
100	900	9-10	90-91	14	10-11	89-90	0,3	0,10	0,3
200	800	19-21	79-81	8	20-22	78-80	0,6	0,1	0,4
300	700	28-31	69-72	16	29-32	68-71	0,5	0,3	0,3
400	600	40-41	59-60	12	41-42	58-59	0,5	0,3	0,2
500	500	48-50	50-52	8	49-51	49-51	0,5	0,4	0,1
600	400	61-62	38-39	10	60-61	39-40	0,7	0,5	0,1
700	300	69-72	28-31	22	68-71	29-32	1,1	0,7	0,4
800	200	79-82	18-21	50	78-81	19-22	1,5	0,8	0,7
900	100	90-92	8-10	60	89-91	9-11	2,6	0,8	1,8
999	1	100	0	96	99	1	10,1	1,2	8,9

Из результатов численных экспериментов (табл. 1–3) можно сделать следующие выводы:

- значения m_1' и m_2' не более чем на единицу отличаются от значений m_1 и m_2, задаваемых формулой (1);

- с ростом числа непрерываемых работ растет доля экспериментов, в которых $\Delta' < \Delta$, а также растут значения Δ, Δ' и $\Delta - \Delta'$.

Литература

1. *Гончаров Е.Н., Кочетов Ю.А.* Вероятностный поиск с запретами для дискретных задач безусловной оптимизации // Дискретный анализ и исследования операций. Сер. 2. 2002. Т. 9. № 2. С. 13-30.

2. *Кочетов Ю.А., Столяр А.А.* Использование чередующихся окрестностей для приближенного решения задачи календарного планирования с ограниченными ресурсами // Дискретный анализ и исследования операций. Сер. 2. 2003. Т. 10. № 2. С. 29-56.

3. *Алексеев О.Г.* Комплексное применение методов дискретной оптимизации. М.: Наука, 1986.

4. *Штовба С.Д.* Муравьиные алгоритмы // ExponentaPro. Математика в приложениях. 2003. № 4(4). С. 70-75.

5. *Glover F., Laguna M.* Chapter 3: Tabu search/ Ed. *R. Colin Reeves*, Modern Heuristics Techniques for Combinatorial Problems. Oxford, Blackwell Scientific Publications, 1993. P. 70-150.

6. *Raghavan R.* Probabilistic Construction of Deterministic Algorithms: Approximating Packing Integer Programs // J. Computer and System Sciences. 1988. V. 37. P. 130-143.

7. *Костенко В.А., Смелянский Р.Л., Трекин А.Г.* Синтез структур вычислительных систем реального времени с использованием генетических алгоритмов// Программирование. 2000. № 5. С. 63-72.

8. *Shen C., Pao Y., Yip P.* Scheduling multiple job problems with guided evolutionary simulated annealing approach // Proc. First IEEE Conf. on Evolutionary Computations. Orlando, 1994. P. 702-706.

9. *Brucker P.* Scheduling Algorithms. Heidelberg, Springer, 2001.

10. *Красовский Д.В.* Алгоритмы решения задачи составления оптимального расписания без прерываний. Диссертация на соискание ученой степени канд. физ.-матем. наук. М.: МФТИ, 2007.

11. *Танаев В.С., Гордон В.С., Шафранский Я.М.* Теория расписаний. Одностадийные системы. М.: Наука, 1984.

12. *Federgruen A., Groenevelt H.* "Preemptive Scheduling of Uniform Machines by Ordinary Network Flow Technique". Management Science, 1986. Vol. 32. № 3. Pp. 341–349.

13. *Gonzales T., Sahni S.* "Preemptive Scheduling of Uniform Processor Systems". Journal of the Association for Computing Machinery, 1978. Vol. 25. № 1. Pp. 92–101.

14. *Буланже Д.Ю., Сушков Б.Г.* Алгоритмы управления вычислительными системами жесткого реального времени. // Изв. АН СССР, Техн. кибернетика, 1982, № 6. С. 160-169.

15. *Буланже Д.Ю.* Оптимальная коррекция директивных интервалов для задачи одного прибора. М.: ВЦ АН СССР, 1983.

16. *Скиндерев С.А., Фуругян М.Г.* Алгоритмы планирования вычислений в многопроцессорных системах с неоднородным множеством работ. М.: ВЦ РАН, 2006.

17. *Гончар Д.Р.* Мультиоценочный алгоритм решения минимаксной задачи составления расписания // Системы управления и информационные технологии, 2007. № 1.3 (27), Москва-Воронеж: Научная книга, 2007. С. 324-328.

ОДИН АЛГОРИТМ РАСПРЕДЕЛЕНИЯ РЕСУРСОВ В МНОГОПРОЦЕССОРНЫХ СИСТЕМАХ С НЕФИКСИРОВАННЫМИ ПАРАМЕТРАМИ

Е.О. Косоруков, М.Г. Фуругян

Рассматривается 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. *Работа выполнена при частичной финансовой поддержке гранта Президента РФ по поддержке ведущих научных школ НШ – 4806.2010.1.*

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 всех работ $i \in N$ установлен общий директивный интервал $(0, F]$ (т.е. работы могут 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

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

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

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:

1) при фиксированном директивном сроке F найти минимальную величину ресурса R и распределение ресурса $(r_1^0, r_2^0, ..., r_n^0)$, при которых существует допустимое расписание (т.е. такое расписание, при котором каждая работа выполняется в своем директивном интервале);

2) при фиксированной величине ресурса R найти минимальный директивный срок F и распределение ресурса $(r_1^0, r_2^0, ..., r_n^0)$, при которых существует допустимое расписание;

3) найти оптимальное сочетание величины ресурса R и директивного срока F, при которых существует допустимое расписание. При этом распределение ресурса $(r_1^0, r_2^0, ..., r_n^0)$ должно удовлетворять ограничениям (1), (2).

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 [2], а задача с произвольными процессорами и одинаковыми директивными интервалами – в [3]. 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 [4] и была сведена к задаче нелинейного программирования. Задача с дополнительным ресурсом и произвольными директивными интервалами рассматривалась в [5] и была сведена к задаче о потоке минимальной стоимости.

2. Задача с фиксированным директивным сроком

Предположим, что величины R и F фиксированы. Как следует из [1], необходимым и достаточным условием существования допустимого расписания в рассматриваемой задаче является выполнение неравенства $\sum_{i\in N} t_i \le mF$, или

$$\sum_{i\in N}(d_i - a_i r_i) \le mF, \quad \sum_{i\in N} a_i r_i \ge \sum_{i\in N} d_i - mF.$$

Пусть $\sum_{i\in N} d_i - mF = B$. Тогда задача заключается в поиске такого распределения ресурса (r_1, r_2, …, r_n), которое удовлетворяет системе ограничений

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

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

Определим минимальное количество R_{min} дополнительного ресурса, при котором для заданного директивного срока F допустимое расписание существует. Рассмотрим задачу линейного программирования:

$$
\begin{aligned}
R &\to \min_{(r_1,\ldots,r_n,R)}, \\
\sum_{i\in N} a_i r_i &\geq B, \\
\sum_{i\in N} r_i &\leq R, \qquad (4) \\
r_i &\in [0,\ \bar{r}_i],\ i = 1, 2, \ldots, n, \\
R &\geq 0.
\end{aligned}
$$

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

В дальнейшем будем рассматривать следующую модификацию задачи (4):

$$
\begin{aligned}
R &\to \min_{(r_1,\ldots,r_n,R)}, \\
\sum_{i\in N} a_i r_i &\geq B, \\
\sum_{i\in N} r_i &= R, \qquad (5) \\
r_i &\in [0,\ \bar{r}_i],\ i = 1, 2, \ldots, n, \\
R &\geq 0.
\end{aligned}
$$

3. Задача с фиксированным количеством ресурса

Будем предполагать, что теперь количество ресурса R в системе фиксировано, а директивный срок может изменяться, за счет чего будет достигаться существование допустимого расписания. В этом случае решается следующая задача линейного программирования:

$$
\begin{gathered}
F \to \min_{(r_1,\ldots,r_n,F)}, \\
\sum_{i\in N} a_i r_i + mF \geq \sum_{i\in N} d_i, \\
\sum_{i\in N} r_i \leq R, \\
r_i \in \left[0, \overline{r_i}\right], i \in N.
\end{gathered}
\tag{6}
$$

Задача (6) имеет решение. Каждому ее решению $(r_1^0, r_2^0, \ldots, r_n^0)$, $F_{\min}$ соответствует допустимое расписание, а $F_{\min}$ – минимально допустимый директивный срок при фиксированной величине ресурса R в системе. Применив алгоритм упаковки [1], получим окончательное решение в виде допустимого расписания.

В дальнейшем будем рассматривать следующую модификацию задачи (6):

$$
\begin{gathered}
F \to \min_{(r_1,\ldots,r_n,F)}, \\
\sum_{i\in N} a_i r_i + mF = \sum_{i\in N} d_i, \\
\sum_{i\in N} r_i \leq R, \\
r_i \in \left[0, \overline{r_i}\right], i \in N.
\end{gathered}
\tag{7}
$$

4. Задача с двумя нефиксированными параметрами

Будем рассматривать теперь случай, когда нефиксированными являются как общее количество ресурса, так и директивный срок. Необходимость рассмотрения этой задачи связана с тем, что величины ресурса и директивного срока, получаемые при решении задач (4) и (6) (или (5) и (7)) соответственно, могут получиться недопустимо большими для тех или иных приложений. Поэтому для более приемлемого результата

следует определить целевую функцию, зависящую сразу от двух значимых параметров (величины ресурса и директивного срока) и обеспечивающую некоторую сбалансированность этих показателей.

Пусть F_0 – значение директивного срока F в задаче (5), R^* – минимальное значение функционала R в этой задаче, R_0 – значение величины ресурса R в задаче (7), F^* – минимальное значение функционала F в этой задаче. Рассмотрим следующую функцию от двух переменных r, f :

$$H_\alpha(r,f)=\alpha\left[\frac{r}{R_0}-1\right]+(1-\alpha)\left[\frac{f}{F_0}-1\right], \qquad (8)$$

где, α - параметр, регулирующий предпочтение между ресурсом и директивным сроком в системе. Множество допустимых значений переменных – интервалы $r\in[R_0,R^*), \quad f\in[F_0,F^*)$.

Рассмотрим следующую задачу:

$$\begin{gathered} H_\alpha(r,f)\to\min_{(r_1,\ldots,r_n,r,f)},\\ \sum_{i\in N}a_i r_i+mf\ge\sum_{i\in N}d_i,\\ \sum_{i\in N}r_i\le r,\\ r_i\in[0,\bar{r_i}],i\in N. \end{gathered} \qquad (9)$$

Будем искать решение этой задачи в следующем виде:

$$r=(1+\gamma)R_0, \qquad (10)$$

$$f=(1+\beta)F_0 \qquad (11)$$

В свою очередь, величины R^*, F^* также могут быть представлены аналогичным образом, а именно, $R^*=(1+\gamma^*)R_0$, $F^*=(1+\beta^*)F_0$. Подставив выражения для r и f из формул (10),

(11) в (8), (9), получим целевую функцию $H_\alpha(\beta,\gamma)=\alpha\gamma+(1-\alpha)\beta$ и ограничения

$$\begin{gathered}\sum_{i\in N} a_i r_i + mF_0 + m\beta F_0 \geq \sum_{i\in N} d_i,\\ \sum_{i\in N} r_i \leq R_0 + \gamma R_0.\end{gathered} \tag{12}$$

4.1. Частное решение

Вернемся к задачам (5) и (7).

Задача (5). Пусть решением этой задачи является пара $\left(R^*,\langle r_R\rangle\right)$, где $\langle r_R\rangle=(r_{R1},...,r_{Rn})$ - распределение ресурсов. Перепишем (5) в следующем виде:

$$\begin{gathered}R\to\min_{(r_1,...,r_n,R)},\\ \sum_{i\in N} a_i r_i + mF_0 \geq \sum_{i\in N} d_i,\\ \sum_{i\in N} r_i = R,\\ r_i\in\left[0,\overline{r_i}\right], i\in N.\end{gathered}$$

Поскольку решением этой задачи является пара $\left(R^*,\langle r_R\rangle\right)$, то с учетом (10), (11) получаем:

$$\begin{gathered}\sum_{i\in N}\langle r_R\rangle_i = R_0 + \gamma^* R_0,\\ \sum_{i\in N} a_i\langle r_R\rangle_i + mF_0 \geq \sum_{i\in N} d_i.\end{gathered} \tag{13}$$

Задача (7). Пусть решением этой задачи является пара $\left(F^*,\langle r_F\rangle\right)$, где $\langle r_F\rangle$- вектор распределения ресурсов. Задача 7 имеет вид:

$$F \to \min_{(r_1,\ldots,r_n,F)},$$
$$\sum_{i\in N} a_i r_i + mF = \sum_{i\in N} d_i,$$
$$\sum_{i\in N} r_i \le R_0,$$
$$r_i \in \left[0, \overline{r_i}\right], i \in N.$$

Подставляя решение $\left(F^*, \langle r_F \rangle\right)$ этой задачи в систему ограничений задачи (7), получаем:

$$\begin{gathered}\sum_{i\in N} \langle r_F \rangle_i \le R_0,\\ \sum_{i\in N} a_i \langle r_F \rangle_i + mF_0 + m\beta^* F_0 = \sum_{i\in N} d_i.\end{gathered} \tag{14}$$

Выразим из (13) γ^*, а из (14) β^*:

$$\begin{gathered}\gamma^* = \frac{\sum_{i\in N} \langle r_R \rangle_i - R_0}{R_0},\\ \beta^* = \frac{\sum_{i\in N} d_i - \sum_{i\in N} a_i \langle r_F \rangle_i - mF_0}{mF_0}.\end{gathered} \tag{15}$$

Лемма 1. Решение $\langle r \rangle$ задачи (9) удовлетворяет следующему условию:

$$\sum_{i\in N} \langle r_F \rangle_i \le \sum_{i\in N} r_i \le \sum_{i\in N} \langle r_R \rangle_i \tag{16}$$

Доказательство. Выпишем ограничения задачи (3):

$$\sum_{i\in N} a_i r_i + mF_0 + m\beta F_0 \ge \sum_{i\in N} d_i, \quad \sum_{i\in N} r_i \le R_0 + \gamma R_0$$

Отсюда следуют оценки:

$$\gamma \geq \frac{\sum_{i \in N} r_i - R_0}{R_0}, \qquad \beta \geq \frac{\sum_{i \in N} d_i - \sum_{i \in N} a_i r_i - mF_0}{mF_0}. \tag{17}$$

Поскольку $\gamma \leq \gamma^*$ и $\beta \leq \beta^*$, то из (15) получаем неравенства (16). Лемма доказана.

Теперь найдем одно из возможных значений вектора r. Предположим, что

$\langle r_F \rangle = \left(\langle r_F \rangle_1, \langle r_F \rangle_2, ..., \langle r_F \rangle_n\right)$ и $\langle r_R \rangle = \left(\langle r_R \rangle_1, \langle r_R \rangle_2, ..., \langle r_R \rangle_n\right)$.

Рассмотрим следующие решения:

1. $r_i = \begin{cases} \dfrac{\langle r_R \rangle_i + \langle r_F \rangle_i}{2}, \text{если } \langle r_F \rangle_i \geq \langle r_R \rangle_i \\ \langle r_F \rangle_i, \text{если } \langle r_F \rangle_i \leq \langle r_R \rangle_i \end{cases}$

(решение, смещенное к $\langle r_F \rangle$);

2. $r_i = \begin{cases} \dfrac{\langle r_R \rangle_i + \langle r_F \rangle_i}{2}, \text{если } \langle r_R \rangle_i \geq \langle r_F \rangle_i \\ \langle r_R \rangle_i, \text{если } \langle r_R \rangle_i \leq \langle r_F \rangle_i \end{cases}$

(решение, смещенное к $\langle r_R \rangle$);

3. $r_i = \dfrac{\langle r_R \rangle_i + \langle r_F \rangle_i}{2}$ (распределенное решение);

4. $r_i^S = \begin{cases} \langle r_F \rangle_i, \text{с вероятностью } q_i \\ \langle r_R \rangle_i, \text{с вероятностью } 1 - q_i \end{cases}$ (стохастическое решение). В качестве q_i можно взять, например, следующую величину:

$$q_i = \begin{cases} \dfrac{\langle r_F \rangle_i}{\langle r_R \rangle_i}, \langle r_F \rangle_i \le \langle r_R \rangle_i, \\ 1, \langle r_F \rangle_i \ge \langle r_R \rangle_i. \end{cases}$$

Лемма 2. Стохастическое решение будет удовлетворять неравенству (16) леммы 1.

Доказательство. Пусть $N = \{1,...,n\}$ и $N = N_1 \cup N_2$, где

$$q_i = \begin{cases} \langle r_F \rangle_i \le \langle r_R \rangle_i, & i \in N_1, \\ \langle r_R \rangle_i \le \langle r_F \rangle_i, & i \in N_2. \end{cases}.$$

Будем интерпретировать значение r_i в виде математического ожидания случайной величины r_i^S, а именно

$$r_i = \langle r_F \rangle_i \cdot q_i + \langle r_R \rangle_i \cdot (1 - q_i), \text{ где } q_i = \begin{cases} \dfrac{\langle r_F \rangle_i}{\langle r_R \rangle_i}, & i \in N_1 \\ 1, & i \in N_2 \end{cases}.$$

Если $i \in N_1$, то

$$r_i = \frac{\langle r_F \rangle_i^2}{\langle r_R \rangle_i} + \langle r_R \rangle_i - \langle r_F \rangle_i \le \frac{\langle r_F \rangle_i^2}{\langle r_F \rangle_i} + \langle r_R \rangle_i - \langle r_F \rangle_i = \langle r_R \rangle_i \quad \text{и}$$

$$r_i = \frac{\langle r_F \rangle_i^2}{\langle r_R \rangle_i} + \langle r_R \rangle_i - \langle r_F \rangle_i = \langle r_F \rangle_i + \frac{(\langle r_R \rangle_i - \langle r_F \rangle_i)^2}{\langle r_R \rangle_i} \ge \langle r_F \rangle_i.$$

Если $i \in N_2$, то $r_i = \langle r_F \rangle_i$. Таким образом,

$\sum_{i \in N} r_i = \sum_{i \in N_1} r_i + \sum_{i \in N_2} \langle r_F \rangle_i$. Отсюда следует, что

$$\sum_{i\in N_1} r_i + \sum_{i\in N_2} \langle r_F \rangle_i \geq \sum_{i\in N_1} \langle r_F \rangle_i + \sum_{i\in N_2} \langle r_F \rangle_i = \sum_{i\in N} \langle r_F \rangle_i,$$

$$\sum_{i\in N_1} r_i + \sum_{i\in N_2} \langle r_F \rangle_i \leq \sum_{i\in N_2} \langle r_F \rangle_i + \sum_{i\in N_1} \frac{\langle r_R \rangle_i + \langle r_F \rangle_i}{\langle r_F \rangle_i - \langle r_R \rangle_i} \leq \sum_{i\in N} \langle r_R \rangle_i.$$

На основании этих решений, получаются значения γ и β. Затем вычисляется штраф H_α и в качестве ответа выдается минимальное из получившихся значений.

Литература

1. *Танаев В.С., Гордон В.С., Шафранский Я.М.* Теория расписаний. Одностадийные системы. М.: Наука, 1984.

2. *Federgruen A., Groenevel H.T.* Preemptive scheduling of uniform machines by ordinary network flow technique// Management Science, 1986, 32, 3, pp. 341 –349.

3. *Gonzales T., Sahni S.* Preemptive scheduling of uniform processor systems//Journal of the Association for Computing Machinery, 1978, 25, 1, pp. 92 – 101.

4. *Давыдов Э.Г.* Исследование операций. М.: Высшая школа, 1990.

5. *Фуругян М.Г., Косоруков Е.О.* Некоторые алгоритмы распределения ресурсов в многопроцессорных системах. Вестник МГУ, сер. 15, 2009, № 4, с. 34–37.

АЛГОРИТМЫ ОРГАНИЗАЦИИ КОНТРОЛЯ В СИСТЕМАХ РЕАЛЬНОГО ВРЕМЕНИ

М.Г. Фуругян

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.

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 увели-

чению 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 [1 - 13]. В работах [1, 2] предполагалось, что модули-буфера надо расставлять равномерно, и получена оптимальная частота их расположения. В работе [1] вероятность возникновения сбоя или ошибки предполагалась малой, а в работе [2] величина оптимального интервала между последовательными модулями-буферами была найдена приближенно. Точная формула для длины оптимального интервала была получена в работах [3, 4]. В работе [5] рассматривается неравномерное расположение модулей-буферов для случая, когда вероятность возникновения ошибки не является постоянной. В работах [6, 7] 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 [8] предлагается эвристическая расстановка модулей-буферов, основанная на увеличении интервала между последовательными модулями-буферами, если некоторое время не было ошибки. В работе [9] 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 [10] был применен метод стохастического динамического программирования для вычисления оптимального расположения модулей-буферов между задачами заданной длины. В [11] рассмотрена многопроцессорная система и получено расположение модулей-буферов, при котором вероятность выполнить работу до выхода из строя всех процессоров максимальна. В [12] получен алгоритм динамической расстановки модулей-буферов во время работы системы в зависимости от изменения обстоятельств. В работе [13] 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 невозможно 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 [14].

Во-вторых, в работах [1 - 13] 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 [15]. 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 [15] строится зона рестарта – минимальная область программы, подлежащая перезапуску после обнаружения ошибки некоторым модулем контроля. Именно эта модель взята за основу в настоящей работе. В [15], однако, предполагается, что модули контроля и модули-буфера уже расставлены некоторым образом.

В настоящей работе рассматривается задача оптимальной расстановки модулей контроля и модулей-буферов в заданном

графе, вершинами которого являются рабочие модули. Рассмотрены случаи, когда граф зависимости по данным является цепочкой (последовательное выполнение работ). Более общие случаи будут опубликованы позднее.

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, $i = 1, 2, \ldots, k$, которые проверяют рабочие модули на наличие ошибок, и модули-буфера B_i, $i = 1, 2, \ldots, m$, 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. Кроме того, предполагается, что выполнены следующие условия.

1. Граф G – это цепочка из n вершин, т.е. рабочие модули $M_1, M_2, \ldots, M_n$ 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.

3. Предварительные результаты.

В этом разделе мы сформулируем некоторые предварительные результаты, которые понадобятся в дальнейшем при построении оптимальной расстановки модулей контроля и модулей-буферов. Доказательство этих утверждений содержится в [16]. 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 $k \geq m$. 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 $k \geq m$ 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,..., C_k и модули-буфера B_1, B_2,..., 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 кон-

троля 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. Случай $k = m$.

Согласно теореме 1 можно без ограничения общности считать, что количество модулей-буферов не превосходит количество 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.2 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 модули 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 в разд. 2 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

$$M_{onm}(k,k,n) = n + \alpha \cdot k(n/k)^2 = n + \alpha n^2 / k.$$

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

$$M_{опт}(k,1,n) = n + a \cdot n^2 \frac{(k+1)}{2k}. \qquad (3)$$

5. 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. 2 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

$$t = n \cdot \frac{y+1}{2y} : \left(\frac{x+1}{2x} + \frac{y+1}{2y} \right) = n \cdot \frac{x(y+1)}{2xy + x + y};$$
$$n - t = n \cdot \frac{y(x+1)}{2xy + x + y}. \qquad (5)$$

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 $b = 1$, $x = 1$, $y = 2$, $t = \dfrac{3}{7}n$, $n - t = \dfrac{4}{7}n$. Оптимальная расстановка модулей контроля и модулей-буферов следующая. Модуль-буфер B_1 располагаем вначале цепочки всех модулей, а B_2 – после $\left[\dfrac{3}{7} \cdot n\right]$ рабочих модулей. Модуль контроля C_1 располагается вместе с модулем B_2, C_2 после $\left[\dfrac{5}{7} \cdot n\right]$ 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

$$M_{опт}(3,2,n) = n + \frac{a \cdot n^2}{4} + \frac{a \cdot n^2}{2 \cdot 3} + \frac{a \cdot n^2}{2 \cdot 3(3^2 + 2 \cdot 3 - 1)} = n + \frac{3}{7} \cdot a \cdot n^2$$

6. Случай произвольного m.

В этом разделе мы построим оптимальную расстановку для общего случая, при условии выполнения предположений 1 – 3 разд. 2. Имеется цепочка из n последовательных рабочих модулей $M_1, M_2,\ldots, M_n$, k модулей контроля $C_1, C_2,\ldots, C_k$ и m модулей-буферов $B_1, B_2,\ldots, B_m$. 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 $C_{b_1+b_2}$, …, модуля-буфера B_m – с модулем контроля $C_{b_1+b_2+\ldots+b_{m-1}}$. Определим также $b_m = k - \sum_{i=1}^{m-1} b_i$. Пусть модули-буфера разбивают цепочку из n рабочих модулей на отрезки $I_1, I_2,\ldots, I_m$, в которых соответственно $t_1, t_2,\ldots, t_m$ рабочих модулей ($\sum_{i=1}^{m} t_i = n$). Согласно теореме 2.3, 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 рассматривать 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 $t_1 \frac{x+1}{2x} = t_m \frac{y+1}{2y} = -\frac{\lambda}{2 \cdot a}$. Обозначим $f(x) = \frac{x+1}{2x}$, то-

гда $t_1 = \frac{f(y)}{f(x)} \cdot t_m$. Подставляя это значение t_1 в третье уравнение системы, получим

$$r \frac{f(y)}{f(x)} \cdot t_m + q t_m = n, \quad t_m = \frac{n}{r \cdot \frac{f(y)}{f(x)} + q} = \frac{f(x) \cdot n}{r f(y) + q f(x)},$$

$$\begin{cases} t_1 = \dfrac{f(b)}{r f(b) + q f(b+1)} \cdot n, \\ t_m = \dfrac{f(b+1)}{r f(b) + q f(b+1)} \cdot n, \end{cases} \tag{8}$$

где $f(x) = \frac{x+1}{2x}, k = mb + r, q = m - r$.

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 $B_1, B_2, \ldots, B_r$ располагаем с интервалами, равными t_1, а $B_{r+1}, B_{r+2}, \ldots, B_m$ – с интервалами, равными t_m. В каждом из полученных отрезков $I_1, I_2, \ldots, I_r$ 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

$$M_{опт}(k,m,n) = n + \frac{a \cdot n^2}{2m} + \frac{a \cdot n^2}{2k} + a \cdot n^2 \frac{r(m-r)}{2km(m(b^2+2b)+r)}, \quad (9)$$

где $k = m \cdot b + r,\ r < m$.

Рассмотрим следующий 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.

Литература

1. *Chandy K.* A survey of analytic models of roll-back and recovery strategies. // Computer 8,5, 1975, pp. 40-47.

2. *Young J. W.* A first-order approximation to the optimum checkpoint interval. // Comm. ACM 17,9, 1974, pp. 530-531.

3. *Gelembe E.* On the Optimum Checkpoint Interval // J. ACM, vol. 26, pp. 259-270 1979.

4. *Duda A.* The Effects of Checkpointing on Program Execution Time. // Information Processing Letters, vol. 16, no. 5, pp. 221-229.

5. *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.

6. *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.

7. *Coffman E., Gilbert E.* Optimal Strategies for Scheduling Checkpoints and Preventive Maintenance, // IEEE Trans. Reliability, vol. 39, no. 1, pp. 9-18, Apr. 1990.

8. *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.

9. *Tantawi A., Ruschitzka M.* Performance Analysis of Checkpointing Strategies. // ACM Trans. Computer Systems, vol. 2, no. 2, pp. 123-144, 1984.

10. *Toueg S., Babaoglu O.* On the Optimum Checkpoint Selection Problem // SIAM J. Computing, vol. 13, no. 3, pp. 630-649, Aug. 1984.

11. *Bruno J.L., Coffman E.G.* Optimal Fault-Tolerant Computing on Multiprocessor Systems // Acta Informatica, vol. 34, pp. 881-904, 1997.

12. *Ecuyer, Malenfant.* Computing optimal checkpointing strategies for rollback and recovery systems // *IEEE Trans. Computers*, **C-37** (4), pp. 491–496.

13. *Tatsuya Ozaki, Tadashi Dohi, Hiroyuki Okamura, Naoto Kaio.* Min-Max Checkpoint Placement under Incomplete Failure Information. // International Conference on DSN, 2004.

14. *Луганская М.И., Сушков Б.Г.* Контроль данных в системах реального времени. // Математические методы управления обработкой информации. М.: МФТИ, 1986. С. 18 – 24.

15. *Белый Д.В., Сушков Б.Г.* Модель организации рестартов в системах реального времени. М.: ВЦ РАН, 1996.

16. *Гречук Б.В., Фуругян М.Г.* Алгоритмы организации рестартов в системах реального времени. М.: ВЦ РАН, 2004.

АЛГОРИТМ СИНТЕЗА МНОГОПРОЦЕССОРНОЙ СИСТЕМЫ

М.Г. Фуругян

Предлагается алгоритм нахождения производительностей процессоров многопроцессорной системы, при которых существует допустимое расписание с прерываниями для заданного множества работ с директивными интервалами.

1. Постановка задачи

Имеется множество работ $N = \{1,\ldots, n\}$, 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, производительности которых равны s_j, $j = 1,\ldots, m$. Работа сложности p_i может быть выполнена на процессоре 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 процессоров, для которых существует допустимое расписание выполнения.

2. Построение области допустимых производительностей процессоров

Пусть $\tau_0 < \tau_1 < \ldots < \tau_k$ - 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 (вы-

полнима) 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.

Как следует из [1, 2], допустимое расписание в рассматриваемой системе существует тогда и только тогда, когда для любого подмножества $N_j \subseteq N$ выполнено неравенство:

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

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

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:

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

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

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

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

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

$$k_1^j S_1 + k_2^j S_2 + \ldots + k_m^j S_m \ge Q^j, \quad j \in J, \tag{2}$$

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

Отметим, что $$S_1 \le S_2 \le \ldots \le S_m. \tag{3}$$

Лемма 1. Неравенство (2) справедливо тогда и только тогда, когда

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

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

Лемма 2. Пусть $S_k = s_1 + s_2 + \ldots + s_k$ $(k = 1, 2,\ldots, m)$. Для того, чтобы по заданным величинам $S_1, \ldots, S_m$ можно было определить величины $s_1, \ldots, s_m$, такие, что $S_1 \ge S_2 \ge \ldots \ge S_m \ge 0$,

необходимо и достаточно выполнение неравенств: $0 \le S_1 \le S_2 \le ... \le S_m$, $2S_r \ge S_{r-1} + S_{r+1}$ ($r = 2, \ldots, m$ -1), $2S_1 \ge S_2$.

При этом

$$s_1 = S_1,\ s_r = S_r - S_{r\text{-}1} \quad (r = 2,\ldots, m). \tag{4}$$

Таким образом, из утверждений лемм 1, 2 следует, что для определения допустимых производительностей процессоров необходимо и достаточно определить величины $S_1,\ldots, S_m$, которые задаются следующими неравенствами:

$$S_m \ge \max_{j \in J} \frac{Q^j}{k_1^j + ... + k_m^j}, \tag{5}$$

$$S_{r+1} \ge S_r \ge \max\left(\max_{j \in J_r} \frac{Q^j - k_m^j S_m - ... k_{r+1}^j S_{r+1}}{k_1^j + ... + k_r^j}, \frac{S_{r+1`} + S_{r-1}}{2} \right), \tag{6}$$

$r = 2, \ldots, m-1,$

$$S_2 \ge S_1 \ge \max\left(\max_{j \in J_1} \frac{Q^j - k_m^j S_m - ... k_2^j S_2}{k_1^j + ... + k_r^j}, \frac{S_2}{2} \right). \tag{7}$$

Для поиска допустимых решений нужно решить систему неравенств (5)-(7) и найти вектор ($S_1, \ldots, S_m$), а затем согласно (4) 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.

Литература

1. *Танаев В.С., Гордон В.С., Шафранский Я.М.* Теория расписаний. Одностадийные системы. - М.: Наука, 1984.

2. *C. Martel.* Preemptive scheduling with release times, deadlines and due times // Journal of the ACM. 1982. V. 29, №3. P. 812-829.

ПРЕДСТАВЛЕНИЕ ГРАФОВ ПРЯМОУГОЛЬНИКАМИ

В.П. Козырев

В работе изучаются два представления графов системой прямоугольников. Определен класс графов, допускающих представление соприкасающимися прямоугольниками. Исследованы φ- и ψ-представления графов. Исследован класс планарных триангуляций. Разработан алгоритм нахождения ψ-представления графов.

1. Основные определения и утверждения

Рассматриваем графы $G(V, E)$ без петель и кратных ребер. Граф G называется планарным, если его вершины можно представить точками евклидовой плоскости и точки $(v_i, u_i) \in E$ соединить жордановыми кривыми так, что они не пересекаются. Известно, что любой планарный граф может быть реализован так, что все эти кривые являются отрезками (теорема Фари). Планарный граф G называется планарной триангуляцией, если каждая его грань является треугольником. Из формулы Эйлера $|V| - |E| + |W| = 2$ вытекает, что в любой триангуляции с n вершинами имеется $3(n\text{-}2) = |E|$ ребер и $2(n\text{-}2) = |W|$ граней. Известно также (теорема Понтрягина-Куратовского), что граф G является планарным тогда и только тогда, когда он не имеет подграфов, гомеоморфных графам K_5 и $K_{3,3}$. Здесь K_r – это полный граф с r вершинами, а $K_{r,s}$ – полный двудольный граф, доли которого содержат r и s вершин. Граф G', гомеоморфный графу G, получается подразбиением некоторых ребер G (может быть всех), т.е. заменой ребер на цепи, содержащие не менее двух ребер.

В работе изучаются два представления графов системой прямоугольников. Каждой вершине $v_i \in V$ ставится в соответствие прямоугольник Π_i (a_i, b_i, c_i, d_i), у которого стороны

(a_i, b_i), (c_i, d_i) параллельны оси O_y, а стороны (a_i, d_i), (b_i, c_i) параллельны оси O_x. В представлении соприкасающихся прямоугольников (φ-представлении) две вершины (v_i, v_j)$\in$ E соединены ребром, если и только если ни один из прямоугольников не принадлежит другому и если одна сторона одного прямоугольника для v_i имеет общий отрезок с какой-то стороной другого прямоугольника для v_j. В другом, более общем представлении прямоугольниками (кратко ψ-представлении) две вершины (v_i, v_j)$\in$ E смежны, если и только если соответствующие прямоугольники пересекаются по площади ненулевой меры. Если для графа G существует первое представление, то для него существует второе представление, но не наоборот. Для доказательства этого заменим каждый отрезок, представляющий смежность вершин прямоугольником, высота которого равна высоте отрезка, а ширина меньше ε/2, где ε - наименьшее евклидово расстояние между парами несоприкасающихся прямоугольников.

Представление графа системой прямоугольников $\Pi_i = \{(a_i, b_i, c_i, d_i), 1 \le i \le n\}$ можно задать $4n$ числами – двумя координатами двух диаметрально противоположных вершин, например, $a_i = (x_i^a, y_i^a)$, $c_i = (x_i^a + \varepsilon_i, y_i^a + \delta_i)$, тогда $b_i = (x_i^a, y_i^a + \delta_i)$, $d_i = (x_i^a + \varepsilon_i, y_i^a)$. Традиционные способы задания графа G с n вершинами и m ребрами требуют: для матрицы смежностей нужно запоминать n (n-1) чисел, принимающих значения 0, 1, для задания списком ребер и списочной структурой, в которой для каждой вершины указываются номера вершин, смежных с ней, нужно запоминать $2m$ чисел. Так, для планарных триангуляций $m = 3(n\text{-}2)$ – нужно запоминать $6(n\text{-}2)$ чисел. Представление графов соприкасающимися прямоугольниками используется при моделировании задач, в которых вершины имеют площади – административные, экономические районы на географической карте, элементы электронных схем (платы). Известны и другие применения ([1]).

2. φ-представления.

Определим класс графов, допускающих представление соприкасающимися прямоугольниками.

Утверждение 1. Любой непланарный граф не является φ-представимым.

Доказывается от противного. Пусть существует φ-представимый непланарный граф G и $\varphi'(G)$ – обратное преобразование, в котором каждый прямоугольник, представляющий вершину графа G, стягиваем в точку и если два из таких прямоугольников имеют общую границу (соприкасаются), соответствующие точки соединяем ребром. В результате получаем изначальный граф G, но он оказывается планарным.

Для формулировки критерия φ-представимости введем необходимые понятия и обозначения. Пусть $L(a, b, V_r)$ – граф, состоящий из смежных вершин a, b, каждая из которых смежна со всеми вершинами множества $V_r = \{v_1, \ldots, v_r\}$, $r \geq 3$ и любые две вершины из V_r не смежны; $M(a, b, V_r)$ – граф, у которого вершина a смежна с вершиной b и всеми вершинами из V_r, $r \geq 4$, на вершинах V_r имеется цикл, вершина b смежна только с вершиной a; K_r – полный подграф с r вершинами, $r \geq 4$.

Теорема 1. Планарный граф G φ-представим тогда и только тогда, когда он не содержит подграфов K_4, $L(a, b, V_3)$ и $M(a, b, V_r)$.

Докажем необходимость. Если граф G имеет один из указанных подграфов, то G не является φ-представимым. Покажем, что граф K_4 не φ-представим. Пусть $V(K_4) = \{v_1, v_2, v_3, v_4\}$. Для любой тройки, например, $\{v_1, v_2, v_3\}$ прямоугольники, их представляющие, должны иметь общую точку, иначе не получим треугольник в графе K_4. Прямоугольники для $\{v_1, v_2, v_4\}$ также должны иметь общую точку и этой точкой не может быть точка p (рис. 1).

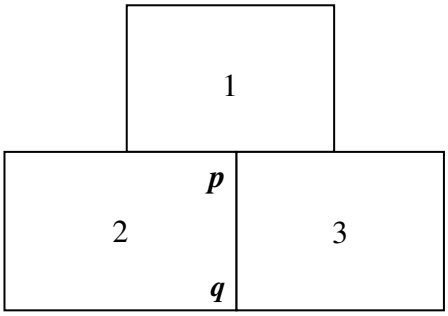
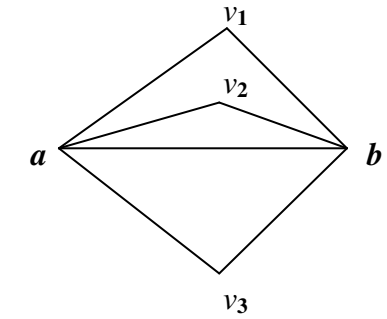


Рис. 1.



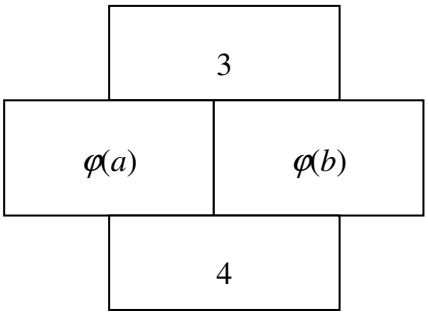


Рис. 2.

Как бы ни разместить на плоскости прямоугольник для вершины v_4, общей точки для троек $\{v_1, v_3, v_4\}$ и $\{v_2, v_3, v_4\}$ не существует. Для графа $L(a, b, V_3)$, он изображен на рис. 2, прямоугольники для v_1 и v_2 должны располагаться по разные стороны относительно пары прямоугольников для a и b, а разместить прямоугольник для v_3, учитывая треугольник (a, b, v_3) в G, негде.

Вершины $a \cup V_r$ графа $M(a, b, V_r)$ представляются следующим образом. Прямоугольники для вершин цикла V_r должны соприкасаться так, чтобы их внутренние стороны образовывали грань как часть плоскости и эта грань есть прямоугольник, который соответствует вершине a. В какой бы грани планарного графа на множестве вершин $a \cup V_r$ не размещалась вершина b, прямоугольник для нее должен размещаться целиком в прямоугольнике $\varphi(a)$ для вершины a, что противоречит определению соприкасающихся прямоугольников. Для $r = 4$ граф на $a \cup V_r$ и его φ-представление изображены на рис. 3. Для $r = 4$ имеем полный граф K_4, который уже не является φ-представимым. Необходимость доказана.

Замечание. Если в определении графа $L(a, b, V_3)$ вершины v_1, v_2, v_3 заменить на связные подграфы, все вершины которых смежны с вершинами a, b, то получаем граф, не являющийся φ-представлением. Подобным образом если в определении $M(a, b, V_r)$ заменить вершину b на связный подграф G', у которого не менее одной вершины соединены ребром с a, то также получаем граф, не являющийся φ-представлением.

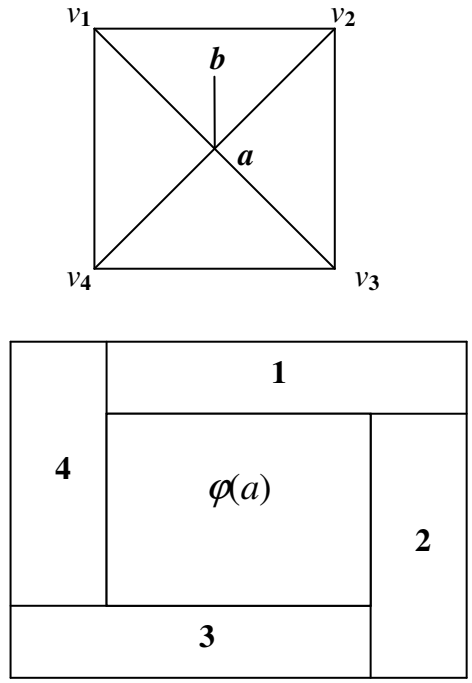


Рис. 3.

Для упрощения доказательства достаточности произведем выделение из начального связного планарного графа 1-связных и 2-связных компонент [2]. Вершина v связного графа G называется точкой сочленения, если при ее удалении граф перестает быть связным и граф $G - v$ состоит из 1-связных компонент. Граф G без точек сочленения называется 2-связным. Анализ соединений точек сочленения позволяет найти подграфы $M(a, b, V_r)$ или показать, что их нет, а анализ связей 2-связных компонент в G позволяет найти подграфы $L(a, b, V_3)$ или доказать, что их нет.

Пусть теперь в связном планарном графе G нет точек сочленения и 2-связных компонент. Отметим, что если для та-

кого графа G существует φ-представление, то любая грань Γ_k с $k \geq 3$ вершинами (и, следовательно, с k ребрами) переходит в грань $\varphi(\Gamma_k)$, ограниченную сторонами прямоугольников, представляющих вершины, ограничивающие Γ_k. Грань Γ_3 не переходит ни в какую грань $\varphi(G)$ (рис. 1). Докажем более сильное утверждение: граф без точек сочленения и 2-связных компонент, не содержащий K_4, допускает φ-представление, в котором каждая грань является прямоугольником.

Доказательство проведем по индукции. База индукции – граф $K_4 \backslash (3,4)$ имеет φ-представление, приведенное на рис. 4.

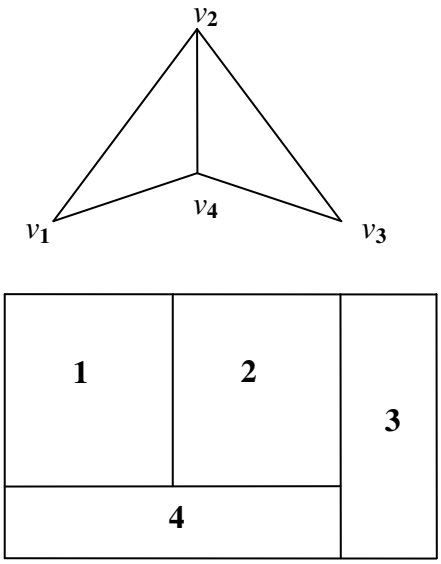


Рис. 4.

Грань (v_1, v_2, v_3, v_4) переходит в грань (1, 2, 3, 3, 3, 4, 4, 1) φ-представление. Удалим из графа G с n вершинами произвольную вершину v_n. Если v_n имеет степень s, то s граней графа G после удаления образуют одну грань и эта грань представляется прямоугольником, имеющим размеры $p \times q$. Пусть

$\varepsilon < \frac{\min(p,q)}{2}$. Увеличим размеры прямоугольников, представляющих вершины, смежные с v_n, на величину ε внутрь образованного прямоугольника и далее впишем прямоугольник, проходящий через измененные стороны. Этот прямоугольник будет представлять вершину v_n в φ (G). Описанное преобразование не может быть применено для случая $s = 3$, в котором вершины, смежные v_n, попарно смежны. Но в этом случае вершина v_n вместе с тремя смежными вершинами, образуют K_4, что исключается по формулировке утверждения о достаточности.

Если вернуться к рассмотрению точек сочленения, не входящих в $M(a, b, V_r)$ и 2-связных компонент, не входящих в образование $L(a, b, V_3)$, то φ-представления, содержащих эти элементы подграфов, присоединяются к любой стороне прямоугольника, вершине, не входящей в прямоугольник графа G. При этом получаем представление G, в котором не все его грани – прямоугольники.

3. Ψ-представление

В ψ-представлении графов с помощью пересекающихся прямоугольников вершины представляются прямоугольниками, как и при φ-представлении, но теперь две вершины смежны, если и только если соответствующие прямоугольники пересекаются по прямоугольнику с ненулевой площадью. В начале статьи было показано, что из φ-представления следует ψ-представление, но не наоборот. Графы K_r, $r \geq 3$; $L(a, b, V_p)$, $M(a, b, V_q)$ допускают ψ-представление. Для K_r это представление получается как множество r различных прямоугольников, каждый из которых содержит некоторый общий прямоугольник. В силу этого определения ψ-представление позволяет использовать вложенные прямоугольники. Это представление допускает графы $L(a, b, V_p)$ и $M(a, b, V_q)$ для любых

$p, q \geq 3$. Полный двудольный граф $K_{n,m}$ для любых $n, m \geq 1$ также допускает ψ-представление, которое приводится на рис. 5.

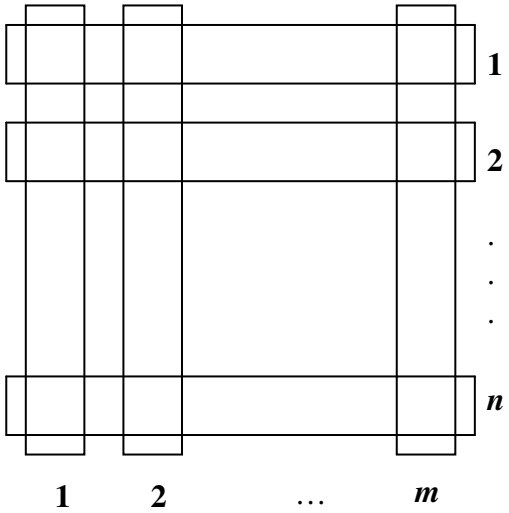


Рис. 5.

Какие же графы не допускают ψ-представление? Решающее значение здесь имеет операция подразбиения ребра, которая заключается в замене ребра (u, v) исходного графа G на цепь $(u, w_1, \ldots, w_k, v)$, $k \geq 1$. Обозначим через $\hat{G}$ граф G, у которого каждое ребро подразбито.*K*

Утверждение 1. Графы $\hat{K}_5$ и $\hat{K}_{3,3}$ не допускают ψ-представление.

Доказательство проведем от противного. Без ограничения общности можем считать, что $k = 1$ для каждой пары u, v. В представлении $\psi(\hat{K}_5)$ и $\psi(\hat{K}_{3,3})$ прямоугольники, представляющие внутренние вершины (вершины подразбиения), не пе-

ресекаются, т.е. не пересекаются ребра графов $\hat{K}_5$ и $\hat{K}_{3,3}$, что противоречит утверждению теоремы Понтрягина-Куратовского.

В завершении первого раздела исследуем важный в теоретическом смысле класс графов – класс планарных триангуляций.

Утверждение 2. Любая планарная триангуляция не допускает φ-представление.

Действительно, внешняя грань любой планарной триангуляции T является треугольником (u, v, w), внутри которого размещаются все остальные (n - 3) вершины. Как бы ни были представлены прямоугольниками эти (n - 3) вершины, «внешние» вершины $\{u, v, w\}$ представить прямоугольниками невозможно.

Утверждение 3. Если в планарной триангуляции T удалено хотя бы одно ребро и в полученном графе G нет подграфов K_4, то G φ-представим.

При удалении одного ребра из T образуется одна грань с 4 вершинами. При удалении k ребер из T может образоваться k граней с 4 вершинами или одна грань с (k + 3) вершинами. Размещаем граф G на плоскости так, чтобы одна из граней, содержащих не менее 4 вершин, стала внешней. Если при удалении k ребер не образуются подграфы $L(a, b, V_3)$ и $M(a, b, V_r)$ и в T нет K_4, то G φ-представим.

Следствие. Для любого непланарного графа G его подразбиение $\hat{G}$ не допускает ψ-представление.

Действительно, в любом непланарном графе G имеется по крайней мере один подграф K_5 или $K_{3,3}$ или их гомеоморфные образы. Производя подразбиение всех ребер G из подграфа K_5 или его гомеоморфного образа получаем граф $\hat{K}_5$, а из подграфа $K_{3,3}$ или его гомеоморфного образа получаем $\hat{K}_{3,3}$.

Оба подграфа $\hat{K}_5$ и $\hat{K}_{3,3}$ не допускают ψ-представление, поэтому и весь граф G его не допускает.

Предполагаем, что граф G ψ-представим тогда и только тогда, когда он не содержит подграфов $\hat{K}_5$ и $\hat{K}_{3,3}$. Для некоторых классов графов гипотеза доказана.

В завершении раздела и статьи опишем алгоритм нахождения ψ-представления, существенным образом основанного на алгоритме установления планарности, который описан в [2]. Осуществляя поиск в глубину изначального графа $G(V, E)$, получаем ориентированный граф $\vec{G}$, состоящий из того же множества вершин V, из дуг корневого остовного дерева, число таких дуг равно $(n - 1)$, и обратных дуг, их число равно $|E| - (n-1)$. Далее переупорядочиваем списки смежностей орграфа $\vec{G}$ так, что дуги, по которым можно попасть в вершину с меньшим номером, используя только одну обратную дугу, стоят в списках раньше. Выбираем в $\vec{G}$ контур C, состоящий из дуг дерева и одной обратной дуги, входящей в вершину 1. Этот контур разбивает плоскость на две области (грани) – внутреннюю и внешнюю. При удалении контура C остающийся граф распадается на сегменты [1]. Теперь размещаем пути, состоящие из пути дерева и одной обратной дуги. Сначала размещаем все пути, принадлежащие сегменту, который начинается в самой высокой вершине (по дереву), и дальше рассматриваются пути, начинающиеся в вершинах ниже. Размещение пути выполняется в трех случаях:

1) начальная и последняя вершины пути принадлежат одной области (грани), и тогда путь размещается внутри этой области и она разбивается на две области (грани);

2) не существует области, содержащей рассматриваемый путь, но существуют две области, имеющие не менее одной общей обратной дуги или путь, содержащий обратную дугу. Одна из этих граней содержит начальную вершину пути, пусть это грань $\Gamma_н$, другая грань (область) содержит последнюю вер-

шину пути, пусть это грань $\Gamma_п$. Такие грани будем называть смежными.

Если начальная или последняя вершина пути смежна с последней или начальной вершиной участка пути, разделяющего эти области, то путь можно разместить.

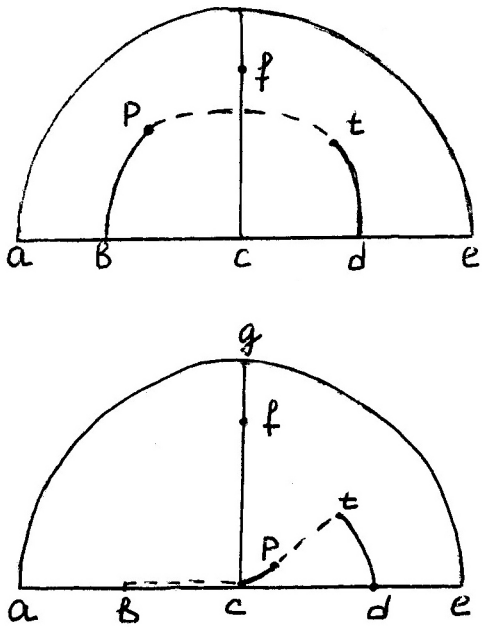


Рис. 6.

Расположение начальных и последних вершин в рассматриваемых путях и формируемых областях приводятся на рис. 6-9.

Пусть *b* или *d* смежна с *c* (для определенности, пусть с *b*), тогда дуга *bp* идет вдоль дуги *bc*, не образуя новой грани. Между *c* и *p* вводим фиктивную дугу и грань *cfgedc* разбивается на две грани: *gfcptdeg* и *cptdc*. Дуга *pc* не учитывается в списках смежностей, а пересечение этой дуги равносильно пересечению дуги *bp*.

3. Не существует смежных граней, как в случае 2, но существует грань (область), одновременно смежная грани $\Gamma_н$ и грани $\Gamma_п$. Обозначим эту промежуточную грань как $\Gamma_{пр}$. Тогда, если начальная вершина пути смежна с путем, разделяющим $\Gamma_н$ и $\Gamma_{пр}$, а конечная смежная с путем, разделяющим $\Gamma_{пр}$ и $\Gamma_п$, размещаем путь в промежуточной грани, разделяя ее на две (рис. 7а и рис. 7б).

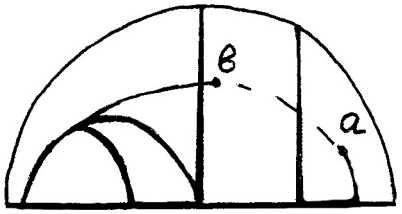


Рис. 7а.

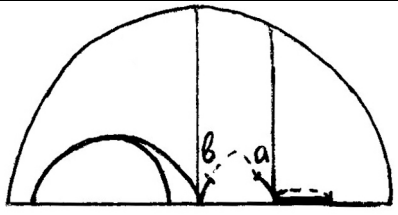


Рис. 7б.

Если бы можно было пересекать большее число граней, то на пути должна существовать вершина, соединенная обратной дугой с внутренним путем *cv*. Но тогда по алгоритму сначала строился бы путь, заканчивающийся этой обратной дугой и следующий путь начинался бы в точке ответвления обратной дуги (см. рис. 8а).

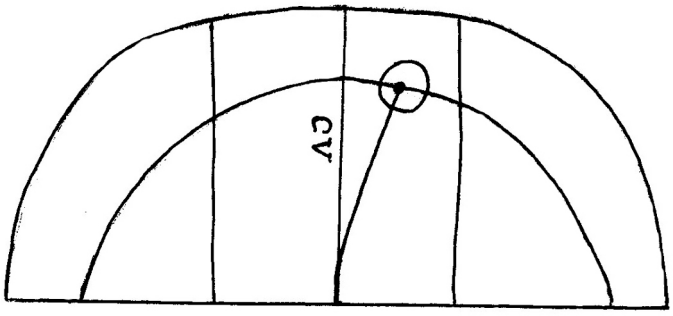


Рис. 8а.

Таким образом, если случаи 1, 2, 3 не реализуются, то нужно удалить все мешающие размещению пути.

Пусть требуется расположить путь, идущий из вершины *s* в вершину *f* (рис. 8б).

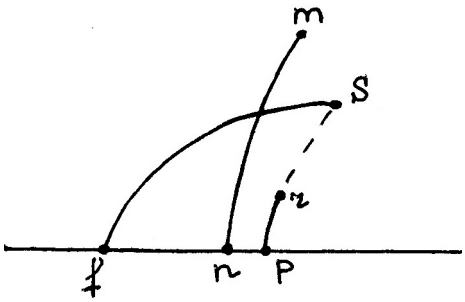


Рис. 8б.

Найдем путь *sp*, заканчивающийся ближайшей к вершине *f* справа от обратной дуги *rp* (среди дуг – потомков *s*). Пусть самый первый путь, мешающий размещению пути *sf* – это путь, заканчивающийся путем *mn*. В этом случае идем к корню сегмента, которому принадлежит этот путь *m, n* до первой вершины *q*, от которой непосредственно начинается обратная дуга. Если эта дуга заканчивается левее вершины *f*, то можно весь подсегмент, начинающийся в вершине *q*, перенести во внешнюю область контура, так как все ребра, инцидентные *q*, могут пересекать цикл *C*.

Если путь *mn* заканчивается правее, то ищем путь, идущий от вершины *q* к самой левой обратной дуге – потомку *q*. Затем находим на этом пути вершину, в которой начинается обратная дуга, заканчивающаяся левее *f* и правее любой другой подобной дуги. Найденные две дуги позволяют выделить подсегмент из сегмента, который можно вынести во внешнюю область. (На рис. 9 подсегмент, заключенный между дугами *a* и *b*, выносится во внешнюю область (грань)).

Если же путь, который пытаемся разместить, принадлежит подсегменту, выносимому наружу, то граф не ψ-представим. Если нет, то выносим подсегмент во внешнюю область и пытаемся разместить пути этого подсегмента во внешней области. Если при этом снова какие-то подсегменты мешают, то переносим их внутрь и так далее, не производя переноса ранее перенесенных подсегментов. Если на каком-то шаге подсегмент нельзя расположить, не трогая ранее расположенных подсегментов, то граф не ψ-представим.

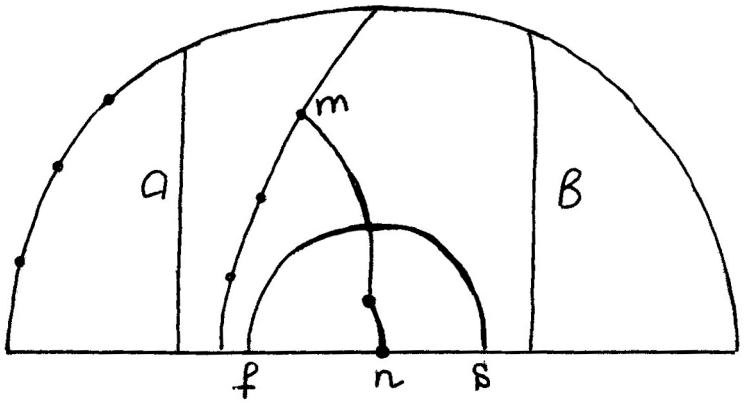


Рис. 9.

Оценим сложность описанного алгоритма.

Максимальное число ребер графа с n вершинами равно $\frac{n^2-n}{2}$. Тогда максимальное число обратных ребер равно

$$N = \frac{n^2-n}{2} - n + 1 = \frac{n^2-3n}{2} + 1 = O(n^2).$$

Число путей равно числу обратных дуг N. При расположении k-го пути мы должны проверить самое большее $k - 1$ путь, причем каждый из них может переноситься только один раз.

Максимальное число путей M, которые нужно просмотреть на всех шагах, равно

$$M = \frac{N^2 - N}{2} + \frac{(N-1)^2 - (N-1)}{2} + ... + \frac{(1-1)^2 - (1-1)^2}{2}$$

Эта сумма имеет порядок $M = O(N^3)$.

Учитывая, что при перемещении одного пути надо просмотреть не более n вершин, получаем сложность L описанного алгоритма:

$L = O(M \cdot n)$.

Подставляя выражения для M и N, получаем оценку

$L = O(N^3 n) = O(n^6 \cdot n) = O(n^7)$.

Такова полиномиальная сложность алгоритма.

Литература

1. *Козырев В.П., Юшманов С.В.* Представление графов и сетей (кодирование, размещения и укладки) // «Итоги науки и техники», сер. теор. вероятности, матем. стат. и теор. кибернетики, изд. ВИНИТИ, 1990, т. 27, с. 129-196.

2. *Рейнгольд Э., Нивергельт Ю., Део Н.* Комбинаторные алгоритмы. Теория и практика. М.: Мир, 1980.

Содержание