

Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра Исследования Операций

Дубровская Анна Викторовна

Жадные алгоритмы составления многопроцессорных расписаний

Бакалаврская работа

Научный руководитель:
д. ф.-м. н., доцент Григорьева Н. С.

Рецензент:
профессор Романовский И. В.

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY

Chair of Operations Research

Anna Dubrovskaja

Greedy algorithms for building multiprocessor schedulings

Bachelor's Thesis

Scientific supervisor:
associate professor N.S. Grigorieva

Reviewer:
professor Joseph Romanovskiy

Saint-Petersburg
2016

Оглавление

1. Введение	4
2. Обзор	5
2.1. Реферат статьи авторов Peter Brucker, M.R.Garey, D.S.Johnson	6
2.2. Реферат статьи авторов M.Y.Kovalyov, Jia Xu.	10
2.2.1. Ветвление.	12
2.2.2. Нижняя граница.	16
2.3. Реферат статьи авторов F.Yalaoui, C.Chu.	18
2.3.1. Теоретическая основа.	18
2.3.2. Алгоритм метода ветвей и границ.	22
3. Постановка задачи.	24
4. Алгоритмы.	26
5. Результаты вычислительных экспериментов.	30
Заключение.	33
Список литературы	34
6. Приложение	36

1. Введение

Задачи упорядочения носят самый общий характер. Они возникают повсюду, где существует возможность выбора той или иной очередности работ: при распределении работ на производстве, составлении расписания приземления самолетов, обслуживании клиентов в банках, формировании очередности выполнения программ вычислительным центром и организации отдыха в праздничные дни. Наша цель — изучить то общее, что характеризует такие задачи независимо от их конкретного содержания.[12]

В терминах теории расписаний можно сформулировать задачу коммивояжера, задачу о разбиении, задачу о раскрое. Идеи и методы, предложенные в теории расписаний, могут быть полезны в других областях дискретной оптимизации. Для решения NP -полных задач часто используются эвристические алгоритмы.

Иногда упорядочение осуществляется случайно, более часто работы выполняются в том порядке, в котором они возникают. Мы не задумываемся о результативности дисциплины ”первый пришел — первый обслужен”, поскольку она отвечает присущему нам чувству естественного порядка. Однако такая дисциплина, может быть, и приемлема при покупке билетов в театральные кассы, но совсем не обязательна для выполнения работ на предприятии.[12]

В приложениях основная трудность упорядочения заключается в том, что возникающие задачи не могут быть расчленены на самостоятельные, не связанные друг с другом части. Более того, они, как правило, увязаны с рядом других проблем, требующих одновременного решения. [12]

Работа состоит из двух частей. Первая часть — реферативная, и содержит теоретическую основу исследования, основанную на статьях П.Брюкера, М.Р.Гэри, Д.С. Джонсона, М.Ю. Ковалева и Дзиа Сю, и Ф. Ялаои и С.Чу. Вторая часть представляет собой исследование жадных алгоритмов: их реализации и тестирование на языке Delphi, результаты экспериментов и выводы.

2. Обзор

В трех статьях, упоминаемых ниже, будет описываться следующая задача P : даны m идентичных процессоров, имеющих скорости $f[L] = 1$, $L = 1, \dots, m$, и множество $T = \{T_1, \dots, T_n\}$ заданий. Для каждого задания дана длина (время выполнения) $c[i]$ и крайний срок $d_i > 0$. Предполагается, что все $d_i, c[i]$ целые. Отношения предшествования задаются с помощью орграфа без циклов G с n вершинами, по одной на каждое задание из T . Граф порождает отношение \prec на T , определяемое так: $T_i \prec T_j$ тогда и только тогда, когда G содержит направленный путь из вершины для T_i в вершину для T_j . В этом случае будем говорить, что T_i предшествует T_j , то есть T_j не может начать выполнение до завершения T_i .

Допустимым расписанием T называется отображение σ из T в неотрицательные вещественные числа, такое что

- a) $|\{T_i \in T : t - 1 \leq \sigma(T_i) < t\}| \leq m \forall t \geq 0$; и
- b) $\sigma(T_i) + 1 \leq \sigma(T_j)$ при $T_i \prec T_j$.

Функция σ назначает время начала выполнения каждому заданию T_i , которое будет выполняться в $[\sigma(T_i), \sigma(T_i) + c[i]]$. Первое условие означает, что не более m заданий могут выполняться одновременно, таким образом будет возможно назначить каждое задание на свой процессор. Второе условие означает, что выполняются отношения предшествования.

Заметим, что крайние сроки d_i могут нарушаться. Функция максимального запаздывания допустимого расписания σ определяется следующим образом

$$L_{\max}(\sigma) = \max\{\sigma(T_i) + 1 - d_i : T_i \in T\}.$$

Если $L_{\max}(\sigma) \leq 0$, будем говорить, что σ удовлетворяет всем крайним срокам, и $L_{\max}(\sigma)$ отражает запаздывания. Если $L_{\max}(\sigma) > 0$, то хотя бы одно задание запаздывающее, и $L_{\max}(\sigma)$ равно максимальному из запаздываний. Для допустимого расписания будет выполнено $L_{\max} \leq 0$.

Один процессор может исполнять лишь одно задание в каждый мо-

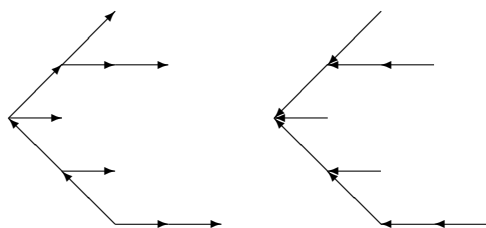
мент времени и каждое задание может исполняться не более, чем на одном процессоре.

2.1. Реферат статьи авторов Peter Brucker, M.R.Garey, D.S.Johnson

”Составление расписаний для заданий единичной длины с древовидными ограничениями предшествования с целью минимизации максимального запаздывания.”

Рассматривается задача $(P|prec = tree, p_i = 1| \max lateness \rightarrow \min)$.

Будут рассматриваться отношения предшествования, заданные направленными деревьями. Рассмотрим два частных случая. Первое — входящие деревья, в которых каждая дуга лежит на пути, направленном к корню. В выходящих деревьях любая вершина лежит на пути, исходящем из корня. Заметим, что граф одного типа может быть сведен к графу другого перенаправлением дуг. На рисунке приведены примеры обоих типов. Для обоих случаев будем писать $T_i \rightarrow T_j$ для обозначения факта наличия в графе дуги из T_i в T_j , то есть того, что T_i непосредственно предшествует T_j .



Целевая функция равна $\max\{\sigma(T_i) + 1 : T_i \in T\}$. Это просто время окончания выполнения последнего по времени задания из σ . Будем считать множеством значений σ натуральные числа.

Алгоритмы для входящих деревьев. Для простоты сначала представлен алгоритм для более простой задачи: даны $m, T, \prec, \{d_i\}$, необходимо найти допустимое расписание, удовлетворяющее крайним срокам, если такое существует. Факт того, что расписание, полученное этим алгоритмом, также минимизирует максимальное запаздывание, будет доказан после того, как будет показано соответствие его упро-

щенной задаче.

Алгоритм состоит из двух основных частей. В первой строится специальное упорядочение заданий из T . Во второй оно используется как приоритетный список для списочного алгоритма. Для каждого целого $t \geq 0$ выбираем задания, предшественники которых уже закончили выполнение. Выбираем как можно больше выполненных заданий в порядке их расположения в приоритетном списке. Формально, полученное допустимое расписание σ имеет $\sigma(T_i)$ распределенным к времени t , к которому должно начать выполнение T_i .

Конкретный приоритетный список L будет иметь следующее свойство: все предшественники задания оказываются перед ним в L . Когда это верно, для входящих деревьев существует эффективный метод (линейный от числа заданий) построения того же расписания, что может быть списочным алгоритмом. Предположим, что T было перенумеровано так, что $L = \langle T_1, T_2, \dots, T_n \rangle$. Этот метод будет назначать начальные времена заданиям в порядке их появления в L и будет использовать вспомогательные переменные. Для каждого i , $1 \leq i \leq n$, переменная $R(i)$ превосходит наибольшее время начала уже назначенных заданий из непосредственных предшественников T_i . Для любого целого момента времени t , $0 \leq t \leq n-1$, переменная $N(t)$ — число заданий, назначенных так, что они начинают выполнение в этот момент. Переменная $FIRST$ дает наименьшее значение t , для которого $N(t) < m$. В начале всем дополнительным переменным присваивается ноль. Затем для каждого i от 1 до n определяется $\sigma(T_i)$ и значения этих переменных обновляются по правилам:

- (1) установим $t \leftarrow \max\{R(i), FIRST\}$;
- (2) установим $\sigma(T_i) \leftarrow t$;
- (3) установим $N(t) \leftarrow N(t) + 1$, если $N(t) \geq m$, $FIRST \leftarrow t + 1$;
- (4) для T_j , непосредственно следующего за T_i , установим $R(j) \leftarrow \max\{R(j), t + 1\}$.

Можно видеть, что в каждом из четырех пунктов достаточно проделать один цикл от 0 до n , а значит, метод строит необходимое расписание за время, ограниченное полиномом от мощности множества T ,

то есть количества заданий, требующих постановки в расписание, $\underline{Q}(n)$. Важно заметить, что так как каждое задание распределяется так, чтобы начать выполнение как можно раньше, и ни одно задание не имеет более одного непосредственно следующего за ним, всегда будет выполнено

$$\{t : N(t) = m\} = \{0, 1, \dots, FIRST - 1\}.$$

Теперь покажем, как строится список L . Основная идея состоит в том, что относительная срочность задания зависит как от его крайнего срока, так и от крайних сроков следующих за ним. Будем устанавливать измененные сроки, учитывающие эти факты. Список L будет построен в соответствии с неубыванием модифицированных крайних сроков.

Наша начальная цель состояла в построении расписания, учитывающего все крайние сроки, если оно существует. Очевидно, если $T_i \prec T_j$, любое допустимое расписание σ , удовлетворяющее всем крайним срокам, должно удовлетворять неравенству

$$\sigma(T_i) + 1 \leq \sigma(T_j) \leq d(j) - 1.$$

То есть, T_i должно быть выполнено не только к d_i , но и к $d_j - 1$. Так как выполнение T_i должно удовлетворять обоим ограничениям, мы можем заменить крайний срок T_i на $\min\{d_i, d_j - 1\}$ без изменения сути задачи. Множество допустимых расписаний остается прежним и расписание удовлетворяет всем крайним срокам в новой задаче тогда и только тогда, когда оно удовлетворяло им в исходной. Действительно, мы можем повторять процесс модификации сроков, пока не получим равносильную задачу со сроками $\{d'_i\}$, для которых $d'_i \leq d'_j - 1$ при $T_i \prec T_j$. Следующий алгоритм строит множество модифицированных сроков и выполняется за время, ограниченное полиномом от количества заданий $\underline{Q}(n)$.

- (1) Для корня T_i входящего дерева, присвоим $d'_i \leftarrow d_i$;
- (2) Выберем задание T_i , для которого еще не назначен модифицированный срок d'_i , но при этом такое, что за ним непосредственно следует T_j , которому он назначен. Если такого T_i нет, конец;

(3) Присвоим $d'_i \leftarrow \min\{d_i, d'_j - 1\}$ и перейдем к шагу 2.

Теорема 1. Пусть $L = \langle T_1, T_2, \dots, T_n \rangle$ приоритетный список, такой что модифицированные крайние сроки $d'_i \leq d'_{i+1}$ для $1 \leq i \leq n - 1$. Тогда допустимое решение, полученное списочным алгоритмом по списку L удовлетворяет исходным срокам тогда и только тогда, когда существует допустимое расписание, удовлетворяющее им.

Таким образом, построен алгоритм построения расписания, удовлетворяющего крайним срокам, если оно существует, состоящего из подсчета модифицированных сроков, составления списка приоритетов, сортировки заданий согласно модифицированным срокам и применения списочного алгоритма к L , он имеет сложность $O(n \log n)$.

Для исходной задачи минимизации наибольшего запаздывания по теореме алгоритм строит допустимое расписание σ с $ML(\sigma) \leq 0$, если оно существует. Он также может быть использован для построения расписания $\sigma : ML(\sigma) \leq l$, если такое существует, применением его к измененной задаче с условиями $m, T, \prec, (d_i + l)$, где l добавляется к каждому крайнему сроку. Если существует допустимое расписание σ для $m, T, \prec, \{d_i\}$ с $ML(\sigma) \leq l$, то это же расписание будет решением новой задачи. Обратно, любое допустимое расписание σ , удовлетворяющее всем крайним срокам в новой задаче будет иметь $ML(\sigma) \leq l$ в исходной.

Списочный алгоритм никак не связан с крайними сроками, поэтому в точности то же расписание строится алгоритмом для измененных задач. В частности, когда l — наименьшее возможное максимальное запаздывание, расписание, построенное алгоритмом будет минимизировать нужное значение запаздывания:

Теорема 2. Пусть $T = \langle T_1, T_2, \dots, T_n \rangle$ приоритетный список, упорядоченный так, что модифицированные сроки удовлетворяют $d'_i < d'_{i+1}$ для $1 \leq i \leq n - 1$. Тогда допустимое расписание, полученное списочным алгоритмом L минимизирует наибольшее запаздывание.

Заметим, что если все исходные крайние сроки одинаковы, алгоритм построит приоритетный список, в котором задания упорядочены по уровню, в соответствии с определением из [8], и поэтому получен-

ное расписание будет минимизировать время обработки. В общем, тем не менее, алгоритм не строит расписание с наименьшим возможным временем обработки среди всех допустимых расписаний, минимизирующий общее запаздывание. Такое расписание может быть найдено повторным применением нашего алгоритма. Пусть l_0 — минимально возможное максимальное запаздывание, определенное при простом применении алгоритма. Для любого целого k , рассмотрим частную задачу, в которой крайний срок каждого задания T_i равен меньшему из значений h и $d_i + l_0$. Допустимое расписание для этой задачи имеет максимальное запаздывание ноль тогда и только тогда, когда его максимальное запаздывание $= l_0$ и время обработки k или меньше для исходной задачи. Можно применить исходный алгоритм несколько раз, в режиме бинарного поиска, для нахождения наименьшего k , для которого полученная задача имеет допустимое расписание с максимальным запаздыванием 0. Расписание, полученное для этого k будет расписанием для исходной задачи, и оно будет иметь наименьшую длину расписания среди всех расписаний с максимальным запаздыванием l_0 . Так как это наименьшее k должно быть целым от n/m до n , требуется хотя бы $O(\log n)$ применений алгоритма, и общая сложность $O(n \log^2 n)$.

2.2. Реферат статьи авторов М.У.Кovalyov, Jia Xu.

”Равномерное планирование с ограничениями на времена освобождения, крайние сроки, отношения предшествования и исключения.”

Бинарное отношение *исключения* определено на множестве задач $T(P)$, так что для любой пары $i, j \in T(P)$ i исключает j , если i и j не могут выполняться одновременно даже на различных процессорах. Каждая задача $i \in S(P)$ имеет время $r[i]$, в которое оно может начать выполнение.

Задача состоит в том, чтобы найти допустимое расписание, то есть такое, что для него выполнены все вышеприведенные требования, и

каждое задание заканчивает свое выполнение до своего крайнего срока, или доказать, что его не существует.

Приложения задачи лежат в планировании до запуска распределения работ в мультипроцессорных системах с жестким временем. Практические примеры включают работу с данными в подводных лодках, расписания доставки авиационного вооружения, контроль движения транспорта, выпуска продукции, робототехнику, управление работой АЭС, и т.д.

Метод ветвей и границ для решения вышеназванной проблемы для случая идентичных процессоров был представлен в статье [4].

Алгоритм, предложенный в статье [4] использует технику метода ветвей и границ. Он имеет дерево поиска, в котором для корневой вершины стратегия, выбирающая в первую очередь наименьший крайний срок, используется для вычисления действующего начального расписания, которое удовлетворяет условиям на времена освобождения и всем отношениям предшествования и исключения.

Для каждой дуги дерева поиска, найдено самое позднее задание l , для которого $L_l = L_{\max}$ в допустимом начальном решении. Определены множества заданий $Z[l]$ и "используемых временных периодов" $UTP[l]$. Соответствующие определения приведены в вышеназванной статье [4].

Утверждается, что если действующее начальное решение не осуществимо, то осуществимое расписание может быть найдено только если возможно перераспределение некоторого сегмента $i \in Z[l]$ так, что некоторый отрезок используемого временного периода $UTP[l]$ занят i . Также если $UTP[l] = \emptyset$, то не существует осуществимого решения для задачи, соответствующей данной дуге.

На основе этих утверждений, если $UTP[l] \neq \emptyset$, то представлены "расширенные множества" $G_1[l]$ и $G_2[l]$, где $G_1[l], G_2[l] \subseteq \{(i, j) | i, j \in Z[l]\}$, и создаются вершины потомков $|G_1[l]| + |G_2[l]|$. Для каждой вершины-потомка, соответствующей паре сегментов $(i, j) \in G_1[l]$, назначается новое отношение — j предшествует i . Для каждой вершины-потомка $(i, j) \in G_2[l]$, назначается отношение j перед i . Это отношение определяется так: если j перед i , то i не может начать выполнение до того,

как начинает выполнение j .

Цель введения таких отношений для всех $i, j \in G_2[l]$ состоит в покрытии всех возможных способов нахождения осуществимых расписаний. Множество $G_1[l]$ и соответствующие отношения предшествования используются для ускорения процесса нахождения осуществимого расписания.

Допустимое начальное решение пересчитывается для каждой вершины-потомка с использованием новых отношений ”перед” и ”предшествует”, и процесс ветвления продолжается до тех пор, пока не построено осуществимое расписание, или ни одна вершина не может быть разветвлена. В последнем случае не существует осуществимого решения для исходной задачи с идентичными процессорами.

Представлены три примера, показывающие, что даже если допустимое начальное решение не осуществимо и $UTP[i] = \emptyset$, осуществимое расписание существует. В примерах показано, что используемые временные периоды и множество $UTP[l]$ не могут быть использованы для определения множества $G_2[l]$.

Ковалев и Сю обнаружили в статье [4] ошибку в определении подходящего задания. Условие $iv)$ в этом определении должно быть прочитано как

$$iv) \neg \exists i : i \text{ EXCLUDES } j \wedge s[i] \leq t \wedge \neg(e[i]) \leq t.$$

Здесь $s[i]$ — начало выполнения задания i .

Если $s[i] < t$ используется в определении, то отношение исключения может не выполняться в допустимом начальном решении, полученном с помощью алгоритма, что влечет к неприемлемому решению исходной задачи. Алгоритм из [4] может быть исправлен удалением требования $UTP[l] \neq \emptyset$ из определения $G_2[l]$.

2.2.1. Ветвление.

Для функции назначения $\alpha : S(P) \rightarrow \{0, 1, \dots, M\}$, если $\alpha(i) = L, L \in \{1, \dots, M\}$, то сегмент i должен быть назначен на процессор

L . Если $\alpha(i) = 0$, то i может быть назначен на любой процессор.

В начале работы алгоритма $\alpha(i) = 0$ для всех $i \in S(P)$.

Назначаются времена высвобождения и крайние сроки соответственно с отношениями предшествования и исключения и значения функции назначения $\alpha(i)$ следующим образом:

Отрегулированное время высвобождения $r'[i, L]$ сегмента i на процессоре L определяется как

$$r'[i, L] = \max\{r[i], A, B, C\},$$

где

$$A = \begin{cases} \min\{r'[j, Q] + c[j]/f[Q] \mid Q = 1, \dots, M\} & \text{если } \exists j : j \prec i, \\ 0 & \text{иначе;} \end{cases}$$

$$B = \begin{cases} r'[j, L] + c[j] & \text{если } \exists j : j \prec i \wedge a(j) = a(i) = L \neq 0, \\ 0 & \text{иначе;} \end{cases}$$

$$C = \begin{cases} \infty & \text{если } a(i) = Q, Q \neq L \wedge Q \neq 0, \\ 0 & \text{иначе.} \end{cases}$$

Отрегулированный крайний срок $d''[i]$ сегмента i определяется как

$$d''[i] = \min\{d[i], D, E\},$$

$$\text{где } D = \begin{cases} d''[j] - c[j] & \text{если } \exists j : i \prec j \\ \infty & \text{иначе;} \end{cases}$$

$$E = \begin{cases} d''[j] - c[j] & \text{если } \exists j : i \prec j \wedge a(i) = a(j) = L \neq 0 \\ \infty & \text{иначе.} \end{cases}$$

В любой момент времени $t, t \in [0, \infty)$, будем говорить, что задание j может быть избран в t на L тогда и только тогда, когда:

- i) $\neg \exists i, i \neq j : i$ выполняется в момент t на L ;
- ii) $t \geq r'[j, L] \wedge \neg \exists L', t' : j$ выполняется в t' на L' ;
- iii) $\neg \exists i : i$ предшествует $j \wedge \neg(e[i] \leq t)$;
- iv) $\neg \exists i : i$ исключает $j \wedge (s[i] < t \wedge e[i] > t) \vee s[i] \geq t \wedge t + c[j]/f[L] > s[i]$ выполняется на
- v) $\neg \exists Q, Q \in \{1, \dots, M\}, Q \neq L : \alpha(j) = Q$.

Это определение гарантирует, что в любой момент t , если задание j

может быть выбрано в момент t на L , то j может быть поставлено на исполнение в t на процессор L , и он будет удовлетворять всем временам высвобождения, отношениям и ограничениям по функции $\alpha(j)$.

В эвристическом алгоритме, подсчитывающем допустимое начальное решение, задачи назначаются в конец текущей последовательности. На каждой итерации определяется множество доступных задач. Из этого множества выбирается задача стратегией $ST \in \{ST1, ST2, ST3\}$. $ST1$ выбирает в первую очередь задачи с наименьшим крайним сроком. Стратегия $ST2$ выбирает задачу с наименьшим значением $d'[j] - c[j]$. Согласно $ST3$, выбирается задача с наименьшим

$$d'[j] - \min\{r'[j, L] + c[j]/f[L] | L = 1, \dots, M\}.$$

Далее выбранная задача назначается на процессор, на котором она может быть исполнена как можно раньше. Расписание строится для каждой стратегии ST . Расписание с наименьшим запаздыванием L_{\max} назначается как допустимое начальное решение.

На каждой итерации подсчитываются значения T_1, \dots, T_M , где T_L — наименьшее значение t , при котором хотя бы одно задание может быть избрано в t на L . Таким образом множество A_L заданий, допустимых к избранию в T_L на процессор L определено. Среди множеств $A_L, L = 1, \dots, M$, задание j выбирается согласно стратегии $ST = ST1$. В случае связей, задание j , такое что $\exists k : j$ предшествует k , выбирается в первую очередь. В случае дальнейших связей, выбирается задание с наименьшим временем выполнения. Определено множество B_j процессоров L , где j может быть избрано в T_L на L . Выбирается тот процессор $L \in B_j$, где j может быть выполнено раньше. Задание j назначается на время с $s[j] = T_L$ до $e[j] = s[j] + c[j]$ на процессоре L . Значения T_1, \dots, T_M исправляются, и указанные выше вычисления повторяются, пока не распределены все задания. Та же процедура используется для $ST = ST2$ и $ST = ST3$.

Заметим, что для построения начального решения может быть использована любая другая разумная стратегия.

Предположим, что допустимое начальное решение не осуществимо.

Пусть l — задание с максимальным запаздыванием из этого расписания. (Если существует более одного задания с максимальным запаздыванием, возьмем в качестве l выполняющееся позднее).

Определим множество заданий $Z[l]$ рекурсивно по следующим правилам:

$$l \in Z[l];$$

$$\forall i, \text{ if } \exists j, L, j \in Z[l], L \in \{1, \dots, M\} :$$

$$\max\{s[i], r'[j, L]\} + c[j] \leq d'[j] \wedge r'[j, L] < e[i], \text{ then } i \in Z[l].$$

Теорема 1. *Предположим, что существует неосуществимое допустимое начальное решение. Если осуществимое расписание существует, то оно имеет одно из следующих свойств:*

- i) некоторое задание $i \in Z[l]$, назначенное на процессор L в допустимом начальном решении, в осуществимом расписании выполняется на процессоре $Q \neq L$;*
- ii) задания $i, j \in Z[l]$, назначенные на один процессор в данном порядке в начальном решении, в осуществимом расписании выполняются на том же процессоре в обратном порядке j, i ;*
- iii) некоторое задание $i \in Z[l]$ в осуществимом расписании выполняется позже, чем в начальном решении.*

Заметим, что требование того, чтобы i выполнялось позднее t , равносильно M условиям: $r'[i, L] = t - c[i] + 1, L = 1, \dots, M$.

Предположим, что допустимое начальное решение не осуществимо. Множество $r^{old}[i, Q] = r'[i, Q], i \in S(P), Q = 1, \dots, M$. Для каждого $i \in Z[l]$, назначенного на процессор L , мы установим следующие новые требования:

- a) $\alpha(i) = Q, Q \in \{1, \dots, M\}, Q \neq L$, если $\alpha(i) = 0$;*
- b) i предшествует j , если j непосредственно предшествует i на L и j не предшествует i ;*
- c) $r'[i, Q] = \max\{r^{old}[i, Q], e[i] - c[i] + 1\}, Q = 1, \dots, M$.*

Для любого $i \in Z[l]$ и любого назначения $a), b), c)$, создается соответствующая вершина-потомок в дереве поиска, пересчитывается зна-

чение $d'[i]$ и $r'[i, L]$ для $i \in S(P)$, $L = 1, \dots, M$ и продолжается процесс ветвления. Теорема 1 показывает, что этот процесс приводит к нахождению осуществимого решения.

2.2.2. Нижняя граница.

Описана процедура подсчета нижней границы запаздывания любого допустимого начального решения.

Сначала рассматривается ослабленная задача. Множество заданий для постановки в расписание $X \subseteq S(P)$, отношения не заданы и $\alpha(i) = 0 \forall i \in X$.

В ослабленной задаче, времена освобождения и крайние сроки сегментов из X переустанавливаются по правилам:

$$\begin{aligned}
 r[v_1] &= \min\{r^{old}[i] \mid i \in X\}, \\
 r[v_2] &= \min\{r^{old}[i] \mid i \in X - \{v_1\}\}, \\
 &\dots \\
 r[v_K] &= \min\{r^{old}[i] \mid i \in X - \{v_1, \dots, v_{K-1}\}\}, \\
 r[i] &= r[v_k], i \in X - \{v_1, \dots, v_k\}, \\
 d[u_1] &= \max\{d^{old}[i] \mid i \in X\}, \\
 d[u_2] &= \max\{d^{old}[i] \mid i \in X - \{u_1\}\}, \\
 &\dots \\
 d[u_k] &= \max\{d^{old}[i] \mid i \in X - \{u_1, \dots, u_{k-1}\}\}, \\
 d[u_i] &= \max\{d[u_k] \mid i \in X - \{u_1, \dots, u_k\}\}.
 \end{aligned}$$

Так как ослабленная задача содержит меньше ограничений, нижняя граница $LB(X)$ для ослабленной границы является также нижней границей и для ослабленной задачи.

Поиск расписания с минимальным запаздыванием для ослабленной проблемы может быть ограничен расписаниями, в которых задания v_L и u_L распределены на процессор L в данном порядке, $L = 1, \dots, K$.

Для такого расписания, пусть C_L — общее время подсчета заданий на процессоре L , то есть $C_L = \sum_{i \in N_L} c[i]$, где N_L — множество заданий процессора L , $L = 1, \dots, k$. Тогда максимальное запаздывание заданий из N_L хотя бы $r[v_L] + C_L - d[u_L]$, и запаздывание расписания может быть оценено

$$\begin{aligned} L_{\max} &\geq \max_{1 \leq L \leq K} \{r[v_L] + C_L - d[u_L]\} = \max_{1 \leq L \leq K} \{C_L - d[u_L] - r[v_L]\} \geq \\ &\geq \max_{1 \leq L \leq K} \{C_L - d[u_L] - r[v_L]\} \geq \frac{\sum_{i \in X} c[i] - \sum_{L=1}^K (d[u_L] - r[v_L])}{K} = LB(X). \end{aligned}$$

Следующая процедура может быть использована для подсчета нижней границы для исходной задачи:

Шаг 1. Присвоим $X = Y = Z[l]$, вычислим $LB(X)$. Установим $LB_1 = LB_2 = LB(X)$. Если $|X| < M$, то перейдем к шагу 2. Если $|X| > M$, перейдем к шагу 3.

Шаг 2. Вычислим $LB(X_i)$ для $X_i = X \cup \{i\}$, $i \in S(P) - X$. Найдем $LB(X_i) = \max\{LB(X_i) \mid i \in S(P) - X\}$. Если $LB(X_j) > LB_1$, то множество $LB_1 = LB(X_j)$, и $X = X_j$. Если $|X| = |S(P)|$, перейдем к шагу 4. Иначе, повторяем шаг 2. Если $LB(X_j) \leq LB$, перейдем к шагу 4.

Шаг 3. Подсчитаем $LB(Y_i)$ для $Y_i = Y - \{i\}$, $i \in Y$. Найдем

$$LB(Y_i) = \max\{LB(Y_i) \mid i \in Y\}.$$

Если $LB(Y_j) > LB_2$, то множество $LB_2 = LB(Y_j)$ и $Y = Y_j$. Если $|Y| = 1$, переходим к шагу 4. Иначе, повторяем шаг 3. Если $LB(Y_j) \leq LB_2$, переходим к шагу 4.

Шаг 4. Подсчитаем

$$LB_3 = \max\{\min\{r'[i, L] + c[i] \mid L = 1, \dots, M\} - d'[i] \mid i \in S(P)\}.$$

Присвоим $LB = \max\{LB_1, LB_2, LB_3\}$.

Авторы использовали этот алгоритм для решения нескольких тестовых задач с двумя единообразными процессорами с числом сегментов

до 10. Во всех случаях, алгоритм не делал разветвление более чем дважды для нахождения осуществимого расписания или определения того, что оно не существует. Для каждого из четырех примеров, представленных в [4], только для получения осуществимого расписания необходимо одно ветвление.

По сравнению с алгоритмом из [4], алгоритм, представленный в этой статье, имеет следующие преимущества:

- могут использоваться различные стратегии нахождения допустимого начального решения. Таким образом увеличивается вероятность начального решения быть осуществимым;

- процесс ветвления включает модификации для ограничений по времени и упорядочивания. Ситуации, в которых изменение только ограничений на упорядочивание не приводит к улучшению структуры допустимого начального решения, могут быть опущены, и будут использованы только временные ограничения;

- процедура подсчета нижней границы является более общей и сильной. Она включает формулы, введенные в [4], как частный случай;

Поэтому можно предполагать, что алгоритм более эффективен в случае идентичных и единообразных процессоров.

2.3. Реферат статьи авторов F.Yalaoui, S.Chu.

Расписания на параллельных машинах для минимизации общего запаздывания.

2.3.1. Теоретическая основа.

Будет решаться задача P : даны m идентичных процессоров, и множество $\{1, \dots, N\}$ заданий. Для каждого задания дана длина (время выполнения) c_i и крайний срок $d_i > 0$. Предполагается, что все d_i, c_i целые.

Отношения предшествования задаются с помощью орграфа без циклов G с N вершинами, по одной на каждое задание. Граф порождает отношение \prec на множестве заданий, определяемое так: $i \prec j$ тогда и

только тогда, когда G содержит направленный путь из вершины для i в вершину для j . В этом случае будем говорить, что i предшествует j , то есть j не может начать выполнение до завершения i .

Заметим, что крайние сроки d_i могут нарушаться. Целевая функция $L(\sigma)$ определяется следующим образом

$$L(\sigma) = \sum_1^N \{t_i - d_i : i \in \{1, \dots, N\}\},$$

где t_i - время окончания выполнения задания i в данном расписании.

Один процессор может исполнять лишь одно задание в каждый момент времени и каждое задание может исполняться не более, чем на одном процессоре. Вытеснения любого выполняемого задания запрещены.

Заметим, что расписание соответствует перестановке множества $\{1, 2, \dots, N\}$. На самом деле, для каждой перестановки соответствующее расписание может быть построено постановкой каждой следующей нераспределенной работы в списке на машину, освобождающуюся раньше других. Обратно, для любого полуактивного расписания, перестановка может быть получена упорядочением работ в порядке неубывания времен начала исполнения. В результате, частичное расписание является перестановкой подмножества из $\{1, 2, \dots, N\}$. Далее понятие расписание может быть истолковано как частичное или полное расписание, если не указано дополнительных предположений. Будут использованы следующие понятия:

$J(K)$: множество работ, распределенных в расписании K ;

$J_m(K)$: множество работ, распределенных на процессор m в расписании K ;

$\Delta_i(K)$: время окончания выполнения работы, непосредственно предшествующей i в расписании K . Если работа i первая в K , $\Delta_i(K)$ считаем равным нулю без потери общности;

$T_m(K)$: общее время запаздывания работ, распределенных на машину m в расписании K ;

$\phi_m(K)$: время выполнения распределенной последней работы на ма-

шине t в расписании K ;

$\mu(K)$: машина, освобождающаяся раньше других в расписании K ;

$K \circ i$: новое расписание, полученное добавлением работы i перед K (то есть добавлением работы i на $\mu(K)$);

$\Sigma(K, i)$: расписание, полученное добавлением к $K \circ i$ частично оптимального расписания оставшихся работ.

При отсутствии неопределенности

$$J(K), J_m(K), \Delta_i(K), C_i(K), T_m(K), \phi_m(K),$$

и $\mu(K)$ можно упростить до $J, J_m, \Delta_i, C_i, T_m, \phi_m$ и μ , соответственно.

Пусть $\mu'(K)$ — машина, свободная сразу после машины $\mu(K)$ и перед всеми остальными машинами системы.

Тогда имеет место утверждение:

Теорема 1. *Существует оптимальное расписание, такое что число работ, распределенных на любую из машин t не превосходит $N - M + 1$.*

Следствие. *Любое частичное расписание K , такое что число работ, распределенных на любую из машин t не превосходит $N - M + 1$, может быть отброшено. В оптимальном расписании, полученном из K , число дополнительных работ на машине t не превосходит $U_m(K) = \min\{N - M + 1 - |J_m(K)|, N - |J(K)|\}$.*

Вдобавок, обозначим через U^* максимум по всем $U_m(K)$; то есть $U^* = \max_{1 \leq m \leq M} U_m(K)$, и также

$$U = U_\mu(K) = \min\{N - M + 1 - |J_\mu(K)|, N - |J(k)|\}.$$

2.2.1.1 Отношения доминирования.

Для задачи минимального суммарного запаздывания на идентичных параллельных машинах получены следующие результаты:

Теорема 2. *Для любого частичного расписания K и любой работы i , нераспределенной в K , если существует другая нераспределенная работа j : $c_j \leq c_i$, $d_j \leq \phi_\mu + c_j$ и $(c_i - c_j)U^* \leq \min\{\phi_{\mu'} - \phi_\mu, d_i - (\phi_\mu + p_i)\}$, то $\Delta(K, i)$ доминируется.*

Теорема 3. Для любого частичного расписания K и любой работы i , нераспределенной в K , если существует другая нераспределенная работа j : $c_j \leq c_i$, $\phi_\mu + c_j \leq d_j$ и $(c_i - c_j)U^* \leq \min\{(d_i - d_j) - (c_i - c_j), (\phi_{\mu'} + c_j) - d_j\}$, то $\Delta(K, i)$ доминируется.

Теорема 4. Для любого частичного расписания K и любой работы i , нераспределенной в K , если другая нераспределенная работа j : $0 \leq c_j - c_i \leq \underline{c}$, $d_j \leq \phi_\mu + c_j \leq d_i$ и $(c_j - c_i)(U - 1) \leq \min\{\phi_{\mu'} - \phi_\mu, d_i - (\phi_\mu + c_i)\}$, то $\Delta(K, i)$ доминируется. Здесь \underline{c} — наименьшее время выполнения из всех нераспределенных работ.

Эти три свойства могут быть доказаны по одной схеме. Изучаются два случая: в первом работы i, j распределяются на один процессор, а во втором на разные. Для обоих случаев, нужно проверить, существует ли другое расписание S не хуже $\Sigma(K, i)$. Такое расписание может быть получено перестановкой позиций работ i и j .

Теорема 5. Для любого частичного расписания K и любой работы i , нераспределенной в K , если другое частичное расписание K' такое что $J(K') = J(K) \cup \{i\}$, $T(K') \leq T(K \circ i)$ и $[N - |J(K)| - 1]\Delta \leq T(K \circ i) - T(K')$, то $\Delta(K, i)$ доминируется. Здесь $\Delta = \max_{k=1}^M (\phi'_k - \phi_k)$, где ϕ_k, ϕ'_k — наименьшие времена выполнения в частичных расписаниях $K \circ i, K'$, соответственно.

2.2.1.2 Нижние границы.

Используя стандартную схему ослабления, трудно найти хорошие нижние границы. Обычно нижняя граница бывает получена ослаблением отношений предшествования. Это связано с тем, что соответствующая задача для одной машины является NP -трудной. Была получена нижняя граница с использованием нового метода MSPTL (Multiple Shortest Processing Time Lists). Без потери общности, предположим, что работы пронумерованы в порядке неубывания времен исполнения $c_1 \leq c_2 \leq \dots \leq c_N$. Строится M однопроцессорных расписаний: для k -го расписания, работы от k до N распределяются в порядке возрастания индексов. Пусть $C_{k,j}$ — j -е наименьшее время исполнения в k -м однопроцессорном расписании. Пусть G — множество этих времен исполнения. Мы отсортируем значения из множества G в порядке возрастания.

Пусть $C_{MSPTL[j]}$ — j -й элемент отсортированного множества. Используя метод *MSPTL*, нижняя граница получается присвоением j -го наименьшего крайнего срока $C_{MSPTL,[j]}$.

Теорема 6. Пусть $(d_{[1]}, d_{[2]}, \dots, d_{[N]})$ — ряд, полученный сортировкой (d_1, d_2, \dots, d_N) в порядке неубывания. Тогда $\sum_{j=1}^N \max(C_{MSPTL,[j]} - d_{[j]}, 0)$ — нижняя граница.

В алгоритме *B&B*, предложенном [6], используется нижняя граница, введенная в работе [5]. Авторы предполагают, что работы пронумерованы в порядке *SPT* и предлагают нижнюю границу равной $\sum_{j=1}^N \max\left\{\frac{\sum_{h=1}^j c_h}{M} - d_{[j]}, 0\right\}$. По *Azizoglu* и *Kirca*, расширяя этот результат на параллельную задачу, предложена следующая нижняя граница для частичного расписания K :

$$\sum_{j \notin J(K)} \max \left\{ \sum_{j=1}^N \left\{ \phi_{\mu}(K) + \frac{c_h}{M} \right\} - d_{|j|}, 0 \right\},$$

где $d_{[j]}$ — наименьший j -й крайний срок нераспределенных задач из K .

2.2.1.3 Верхние границы.

Верхние ограничения генерируются эвристически, используя выбор лучшего решения, полученного после разрешения задачи с различными правилами приоритета. Эти правила *SPT*, *EDD*, *Minimum Slack*. Эта техника является приложением метода, предложенного в статье [1].

2.3.2. Алгоритм метода ветвей и границ.

Алгоритм *B&B* основывается на предложенных ранее в [3] для решения задачи составления расписания на параллельных процессорах с целью минимизации общего времени работы. Каждая вершина представляет собой частичное решение. Сначала нужно вычислить начальное решение, отражающее первую верхнюю оценку. Получено первое решение, как сказано выше, выбором наилучшего решения в результате применения эвристических алгоритмов, основанных на правилах *SPT* (Shortest Processing Times), *EDD* (Earliest Due Date) *Minimum Slack*. Различные стадии работы алгоритма являются фазами распре-

деления работ на машине, освобождающейся раньше других в данном частичном расписании. С этой целью, множество неисследованных вершин упорядочивается в порядке возрастания нижних границ. Для каждой вершины, машины упорядочиваются в порядке неубывания времен освобождения. Перед созданием новой вершины, проверяются ряды отношений доминирования. Для каждой вершины дерева поиска, которая не может быть исключена отношениями доминирования, подсчитывается нижняя граница. Если значение полученной нижней границы не меньше значения текущего решения или значения верхней границы, вершина исключается. Вычислительные эксперименты показали, что оптимальные решения могут быть получены в некоторых случаях с 30 работами на 2 машинах в разумное время. [2]

3. Постановка задачи.

Определено множество заданий $U = \{u_1, \dots, u_n\}$, которые обслуживаются m параллельными идентичными процессорами.

Заданы t_i — времена выполнения работ и D_i — директивные сроки(времена, к которым задания должны быть завершены). d_i — ранний срок начала выполнения работы. $d_i \geq 0$; $t_i, D_i > 0, i = 1, \dots, n, r_i$ — минимально возможное время начало i -й операции.

Составление расписания означает задание сопоставления

$$\forall i \in N (\tau_i, num_i)$$

времен начала исполнения и номеров процессоров, выполняющих данную задачу(для случая с запрещением прерываний).

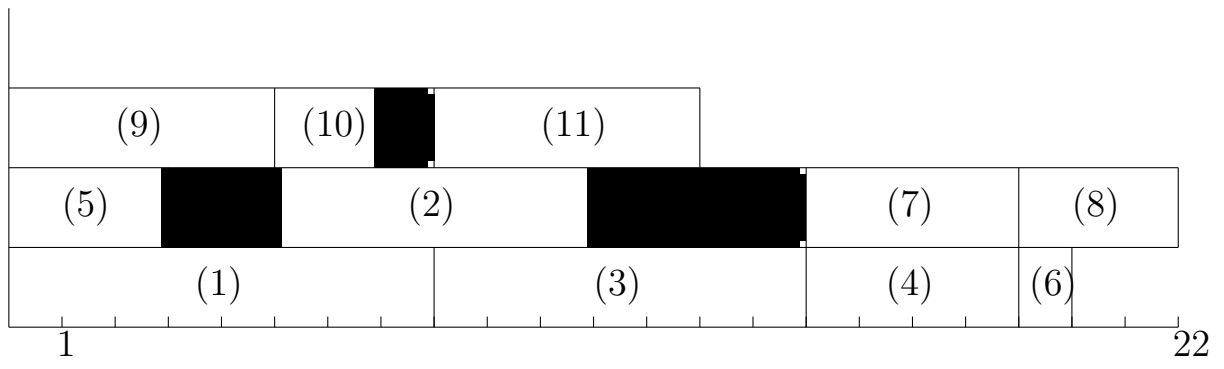
Каждая машина может выполнять лишь одну работу в любой момент времени.[12, 10]

Оптимальным будем называть расписание, соответствующее минимуму функционала(оптимальному значению максимального временного смещения) $L^* = L_{\max}(s) = \max_{i \in N} L_i(s)$. $L_i(s) = \bar{t}_i(s) - D_i = \tau_i + t_i - D_i$ — временное смещение для i -й работы, где $\bar{t}_i(s)$ — момент завершения обслуживания.

Задача построения оптимального расписания сводится к минимизации τ : существует допустимое относительно модифицированных директивных сроков $D'_i = D_i + \tau$ расписание. Оно является оптимальным.

Построение оптимального по быстройдействию расписания $\forall d_i$ сводится к построению расписания с минимальным значением максимального временного смещения при $d_i = 0$. Расписанию, которому соответствует минимальное значение $L_{\max}(s)$, соответствует и минимальное максимальное запаздывание.

На рисунке представлен пример графика Ганта для задачи с тремя процессорами.



4. Алгоритмы.

Жадными (градиентными) называют алгоритмы, действующие по принципу ”максимальный выигрыш на каждом шаге”. Такая стратегия не всегда ведет к конечному успеху — иногда выгоднее сделать не наилучший, казалось бы, выбор на очередном шаге с тем, чтобы в итоге получить оптимальное решение.[9]

Будем рассматривать алгоритмы, различающиеся предпорядкованием для выбора к выполнению работ. Будут представлены 4 варианта:

- задания выполняются в порядке неубывания d ;
- задания выполняются в порядке неубывания $d + D$;
- задания выполняются в порядке неубывания D ;
- задания выполняются в порядке неубывания $D - t$. [11]

Рассмотрим приближенный алгоритм *ELS/IIT*. Этот алгоритм можно отнести к классу жадных алгоритмов. На каждом шаге алгоритма одно из заданий будет окончательно устанавливаться на процессор.

Для каждого задания нам известно раннее начало задания r_i и самое позднее время начала выполнения задания $v_{\max}[u_i] = D_i - t_i$, при котором задание будет закончено до наступления директивного срока. Сначала для постановки в расписание будем выбирать задание с минимальным поздним началом, при этом самые критические задания будут назначаться в первую очередь. Такой подход может приводить к появлению простоев процессоров. Для того, чтобы исключить нерациональное использование времени процессора, будем на время простоя подбирать другое задание, которое можно выполнять в это время, но не увеличивая время начала критического задания. Таким образом выбор задания будет выполняться в два этапа. Пусть k заданий уже поставлено в расписание и частичное расписание S_k построено.

Приближенное решение конструируется алгоритмом *ELS/IIT/d* следующим образом:

1. Найдем процессор l_0 такой, что $t_{\min}(l_0) = \min\{time_k[i] \mid i \in 1 : M\}$;

2. Выберем задание u_0 , такое что

$$v_{\max}[u_0] = \min\{v_{\max}[u_i] \mid u_i \notin S_k\} \& d[u_0] = \min\{d[u_i] \mid u_i \notin S_k\};$$

3. Если $idle(u_0) = r(u_0) - t_{\min}(l_0) > 0$, тогда выберем задание $u^* \notin S_k$, которое может выполняться в период простоя процессора l_0 без увеличения времени начала u_0 , а именно

$$v_{\max}(u^*) = \min\{v_{\max}[u_i] \mid r(u_i) + t(u_i) \leq r(u_0), u_i \notin S_k\};$$

4. Если задание u^* найдено, то назначить на процессор l_0 задание u^* , иначе задание u_0 .

Теперь опишем алгоритм, в котором задания выполняются в порядке неубывания суммы времени поступления задания на выполнение и директивного срока ($d_i + D_i$):

Алгоритм *ELS/IIT/Dd*:

1. Найдем процессор l_0 такой, что $t_{\min}(l_0) = \min\{time_k[u_i] \mid u_i \in 1 : M\}$;
2. Выберем задание u_0 , такое что

$$v_{\max}[u_0] = \min\{v_{\max}[u_i] \mid u_i \notin S_k\} \& (d + D)[u_0] = \min\{(d + D)[u_i] \mid u_i \notin S_k\};$$

3. Если $idle(u_0) = r(u_0) - t_{\min}(l_0) > 0$, тогда выберем задание $u^* \notin S_k$, которое может выполняться в период простоя процессора l_0 без увеличения времени начала u_0 , а именно

$$v_{\max}(u^*) = \min\{v_{\max}[u_i] \mid r(u_i) + t(u_i) \leq r(u_0), u_i \notin S_k\};$$

4. Если задание u^* найдено, то назначить на процессор l_0 задание u^* , иначе задание u_0 .

Далее опишем алгоритм, в котором задания выполняются в порядке неубывания директивных сроков (D_i):

Алгоритм *ELS/IIT/D*:

1. Найдем процессор l_0 такой, что $t_{\min}(l_0) = \min\{time_k[u_i] \mid u_i \in 1 : M\}$;
2. Выберем задание u_0 , такое что

$$v_{\max}[u_0] = \min\{v_{\max}[u_i] \mid u_i \notin S_k\} \& D[u_0] = \min\{D[u_i] \mid u_i \notin S_k\};$$

3. Если $idle(u_0) = r_0 - t_{\min}(l_0) > 0$, тогда выберем задание $u^* \notin S_k$, которое может выполняться в период простоя процессора l_0 без увеличения времени начала u_0 , а именно

$$v_{\max}(u^*) = \min\{v_{\max}[u_i] \mid r(u_i) + t(u_i) \leq r(u_0), u_i \notin S_k\};$$

4. Если задание u^* найдено, то назначить на процессор l_0 задание u^* , иначе задание u_0 .

Ниже опишем алгоритм, в котором задания выполняются в порядке неубывания разности директивных сроков и времен выполнения $(D_i - t_i)$:

Алгоритм *ELS/IIT/dt*:

1. Найдем процессор l_0 такой, что $t_{\min}(l_0) = \min\{time_k[u_i] \mid u_i \in 1 : M\}$;
2. Выберем задание u_0 , такое что

$$v_{\max}[u_0] = \min\{v_{\max}[u_i] \mid u_i \notin S_k\} \& (D - t)[u_0] = \min\{(D - t)[u_i] \mid u_i \notin S_k\};$$

3. Если $idle(u_0) = r(u_0) - t_{\min}(l_0) > 0$, тогда выберем задание $u^* \notin S_k$, которое может выполняться в период простоя процессора l_0 без увеличения времени начала u_0 , а именно

$$v_{\max}(u^*) = \min\{v_{\max}[u_i] \mid r(u_i) + t(u_i) \leq r(u_0), u_i \notin S_k\};$$

4. Если задание u^* найдено, то назначить на процессор l_0 задание u^* , иначе задание u_0 .

В вычислительном эксперименте мы сравним расписания, построенные алгоритмом *ELS/IIT* с расписаниями, построенными алгоритмом без вынужденных простоев *ELS/ND*. Приближенное решение конструируется алгоритмом *ELS/ND* следующим образом:

1. Найдем процессор l_0 такой, что $t_{\min}(l_0) = \min\{time_k[i] \mid i \in 1 : M\}$;
2. Выберем задание u_0 , такое что $v_{\max}[u_0] = \min\{v_{\max}[u_i] \mid u_i \notin S_k\}$;
3. Если $idle(u_0) = r(u_0) - t_{\min}(l_0) > 0$, тогда выберем задание $u^* \notin S_k$, которое может выполняться без простоя процессора s

$$\begin{cases} v_{\max}(u^*) = \min\{v_{\max}[u_i] \mid r(u_i) \leq t_{\min}(u_0) \quad u_i \notin S_k\} \\ d(u) \leq t_{\min} \quad \forall u \end{cases};$$

4. Если задание u^* найдено, то назначить на процессор l_0 задание u^* , иначе назначить на него задание u_0 .

Вычислительная сложность алгоритмов $\underline{O}(n^2)$. [7]

Аналогично могут быть построены алгоритмы для задачи без простоев, выбирающие сначала задания с минимальным поздним началом, директивным сроком, ранним началом, или суммой этих величин, для этого в последнем алгоритме необходимо положить соответственно $v_{\max}[u_i] = D_i - t_i$, $v_{\max}[u_i] = D_i$, $v_{\max}[u_i] = d_i$ или $v_{\max}[u_i] = d_i + D_i$.

5. Результаты вычислительных экспериментов.

Для тестирования алгоритмов были использованы тесты из Standart Task Graph Set, которые доступны на сайте <http://www.kasahara.elec.waseda.ac.jp//schedule>. Standart Task Graph Set — это наборы случайно сгенерированных тестов для проверки алгоритмов составления многопроцессорных расписаний. Для создания исходных данных были использованы тесты, в которых на множестве заданий были заданы отношения предшествования, а в качестве целевой функции рассматривалась длина расписания. Времена выполнения заданий были заданы. Времена поступления заданий и директивные сроки были получены следующим образом. По графу, задающему отношения частичного порядка для множества заданий U , для каждого задания вычислялся самый ранний возможный срок начала задания и брался в качестве времени поступления задания r . Также вычислялось наиболее позднее время окончания задания и выбиралось в качестве директивного срока D . Каждая серия тестов в этом наборе состоит из 180 примеров. Для каждой серии тестов мы построили оптимальные расписания для 2, 4 и 8 процессоров. Были рассмотрены тесты из Standart Task Graph Set для $n = 100$, $n = 300$, где n — количество заданий.

Мы ограничили количество итераций для поиска допустимых решений. Если допустимое расписание S с заданным максимальным временным смещением L для m процессоров не удалось построить за 5000 итераций, предполагалось, что расписание не существует и значение целевой функции L увеличивалось. Такой подход позволил построить расписания для всех тестовых задач, но оставался открытым вопрос, построено оптимальное решение или нет. Результаты вычислительных экспериментов представлены в таблице. Первая колонка содержит тип алгоритма, вторая — число процессоров.

Оптимальные расписания были получены алгоритмом ELS/ND в 19% тестов при $n = 300$ и в 32% тестов при $n = 100$.

Результаты эксперимента для $n = 100$

	m	N_{opt}	$r \in (0, 0.05]$	$r \in (0.05, 0.1]$	$r > 0.1$	t
$D - t$	2	0.57	0.34	0.02	0.07	0.54
	4	0.42	0.34	0.04	0.2	0.36
	8	0.64	0.04	0.02	0.3	1.45
D	2	0.16	0.72	0.04	0.08	0.56
	4	0.11	0.32	0.17	0.4	0.55
	8	0.41	0	0.02	0.57	1.9
$D + d$	2	0.17	0.71	0.04	0.08	0.68
	4	0.12	0.33	0.15	0.4	0.5
	8	0.54	0.01	0.02	0.43	0.15
d	2	0.12	0.71	0.06	0.11	0.74
	4	0.08	0.3	0.15	0.47	0.62
	8	0.44	0	0.01	0.55	0.26

Результаты эксперимента для $n = 300$

	m	N_{opt}	$r \in (0, 0.05]$	$r \in (0.05, 0.1]$	$r > 0.1$	t
$D - t$	2	0.6	0.39	0.01	0	5.2
	4	0.28	0.51	0.02	0.19	6.45
	8	0.45	0.21	0.06	0.28	3.3
D	2	0.12	0.87	0.01	0	6.74
	4	0.01	0.7	0.06	0.23	8.55
	8	0.15	0.11	0.11	0.63	5.1
$D + d$	2	0.1	0.89	0.01	0	7
	4	0.01	0.71	0.06	0.22	8.4
	8	0.28	0.1	0.11	0.51	4.2
d	2	0.14	0.85	0.01	0	8.4
	4	0.01	0.62	0.13	0.24	9.6
	8	0.18	0.06	0.09	0.67	6.2

Оценка $rllbow = r = \frac{f_A - LB}{LB}$.

Колонка N_{opt} показывает долю тестов (в процентах), в которых были получены оптимальные решения. Следующая колонка показывает долю тестов, в которых были получены приближенные решения с оценкой

не более 0.05, но не была доказана оптимальность решений из-за ограничения на число итераций. Но эти решения могут быть и оптимальными. Две следующие колонки показывают долю тестов, в которых $RT \in (0.05, 0.1]$ и RT больше чем 0.1.

Приближенные решения с относительной погрешностью RT менее чем 5% были получены ELS/ND алгоритмом в 64% случаев. Оптимальные решения были получены в среднем примерно в 25.5% случаев.

Также представлен алгоритм для задачи с допущением простоя. Для систем умеренного размера задача может быть решена за разумное время. Результаты экспериментов показали, что "жадный" метод неизменно помогает найти хорошие решения для произвольных систем заданий.

Результаты эксперимента

алгоритм	\bar{x}
d+D	66.2%,
d	58.5%
D	59.7%
D-t	79.8%

"Хорошие" оценки (меньше 0.05) получим в среднем в \bar{x} задач, где средние подсчитаны по формуле

$$\bar{x} = \frac{|\{rlowb_{N_{opt}} + rllowb_{(0,0.05]} | m = 2, 4, 6; n = 100, 300\}|}{6}.$$

Из этих данных можно сделать вывод, что наиболее эффективной для данной задачи показала себя сортировка по $D - t$, за ней следует $D + d$, затем — D , и d .

Заключение.

В этой работе мы рассматривали задачу составления расписания $(P|r_j|L_{\max})$.

В первой части работы были рассмотрены расчеты и алгоритмы, представленные в работах П.Брюкера, М.Р.Гэри, Д.С. Джонсона(1977 года), а также М.Ю. Ковалева и Дзиа Сю, и Ф. Ялаои и С.Чу(около 2000 года). Перевод и реферат работ предложен во втором разделе ”обзор”.

Во второй части работы проводились вычислительные эксперименты для проверки эффективности ”жадного” алгоритма. Для оценки качества решения примем среднее отношения значения решения к нижней границе максимального запаздывания LB . Для демонстрации эффективности мы проверили задачи, сгенерированные случайно, размерностью до 300 заданий. Все они были решены не более чем за 60 секунд.

Оптимальные решения были получены для 63% случаев, полученных ”жадным” методом. Среднее процентное отклонение от нижней границы изменялось в пределах 0.5 — 6.1%.

Список литературы

- [1] A. Dogramaci, J. Surkis // Evaluation of a heuristic for scheduling independent jobs on parallel identical processors. — 1979. — Vol. 25.
- [2] Discrete optimization methods in scheduling and computer-aided design. — Minsk, Belarus, 2000.
- [3] F. Yalaoui, C. Chu. Parallel Machine Scheduling To Minimize Total Completion Time with Release Dates Constraints: An exact Method // Submit for publication to Discrete Appl-d Math.J-l.
- [4] J. Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence and exclusion relations // IEEE Transactions on Software Engineering. — Feb 1993. — Vol. 19.
- [5] Kondakci S., O. Kirca, M. Azizoglu. An efficient algorithm for single machine tardiness problem // Int'l J.of Production Economics. — 1994. — Vol. 36.
- [6] M. Azizoglu, O. Kirca. Tardiness minimization on parallel machines // Int'l J.of Production Economics. — 1998. — Vol. 55.
- [7] N.S.Grigorieva. Branch-and-Bound Method for Scheduling Precedence Constrained Tasks on Parallel Identical Processors. — Proceedings of the World Congress on Engineering, London U.K., July 2-4, 2014.
- [8] T.C. Hu. Parallel sequencing and Assembly Line Problems // Operations Research. — 1961. — Vol. 9.
- [9] В.Е. Алексеев, В.А. Таланов. Графы и алгоритмы. Структуры данных. Модели вычислений. — Москва, Интернет-университет информационных технологий, 2006.
- [10] В.С. Танаев, В.С. Гордон, Я.М. Шафранский. Теория расписаний. Одностадийные системы. — М.: Наука, 1984.
- [11] Н.С. Григорьева. Теория расписаний. — РИО СПбГУ, 1995.

- [12] Р.В. Конвей, В.Л. Максвелл, Л.В. Миллер. Теория расписаний. — Москва, Наука, 1975.

6. Приложение

Программа написана на языке Delphi и представляет себе приложение, получающее на входе файл с данными формата '.stg' и выдающая текстовый файл результата.

Код программы:

```
unit dbegels;
interface
uses dmydata,dbeglast;

procedure ElsInit;
procedure sortdt;
procedure sortd;
procedure sortdd;
procedure sortds;
implementation
  procedure sortd ;    {сортируем по D}
  var x, y, i, j : integer;
  begin
    for j:=0 to npnt do
      begin  y:=3000;
        for i:=0 to npnt do
          if( pnt[i]^tfin + pnt[i]^t < y ) and (pnt[i]^timbeg=-1 ) then
            begin x:=i; y:=pnt[i]^tfin+pnt[i]^t;
              end;
            pedg[j]:=x; pnt[x]^timbeg :=-2;
          end;
        end;
      end;

  procedure sortdt ; {сортируем по D-t}
  var x, y, i, j: integer;
  begin
    for j:=0 to npnt do
      begin  y:=3000;
        for i:=0 to npnt do
          if ( pnt[i]^tfin < y ) and ( pnt[i]^timbeg=-1 ) then
            begin x:=i; y:=pnt[i]^tfin;    end;
            pedg[j]:=x; pnt[x]^timbeg :=-2;
          end;
        end;
      end;
  end;
```

```

procedure sortdd ;    {сортируем по D+d}
var x, y, i, j : integer;
begin
  for j:=0 to npnt do
    begin  y:=3000;
      for i:=0 to npnt do
        if( pnt[i]^tfin + pnt[i]^t + pnt[i]^tor < y ) and
          (pnt[i]^timbeg=-1 ) then
          begin x:=i; y:=pnt[i]^tfin+pnt[i]^t+pnt[i]^tor;
            end;
          pedg[j]:=x; pnt[x]^timbeg :=-2;
        end;
      end;
    end;

procedure sortds ;    {сортируем по d}
var x, y, i, j : integer;
begin
  for j:=0 to npnt do
    begin  y:=3000;
      for i:=0 to npnt do
        if( pnt[i]^tor < y ) and (pnt[i]^timbeg=-1 ) then
          begin x:=i; y:=pnt[ i]^tor;
            end;
          pedg[j]:=x; pnt[x]^timbeg :=-2;
        end;
      end;
    end;

procedure ElsInit;
var i, j: integer;
begin min:=10000;
  {считаем поздние начала}
  for i:=1 to s do time[i]:=0;
  for i:=0 to npnt do
    begin
      pnt[i]^res:=0; pnt[i]^timbeg:=-1; pnt[i]^nproc:=0;
      plan[i]:=nil; prost[i]:=0;
      for j:=0 to npnt do
        g[i]:=0;
    end;
end;

```

```

        end;
end;
begin end.

unit dbeglast;
{процедура нужна для получения времен поступления и директивных сроков,
так как исходные данные формируются из графа}
interface
uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, dmydata;
procedure vertout(vrt:ppoint);
procedure begtime (var f: text;npnt,nedg:integer;vert:integer);
procedure lasttime(var f: Text;npnt,nedg: integer;vert,tpt:integer);
Implementation
procedure vertout;
var vrt1:ppoint; lsp : list; i:Integer;
begin with vrt^ do
    begin
        lsp:=last;
        while (lsp <> nil) do
            begin i:= lsp^.ipoint;
                vrt1:=pnt[i];
                with vrt1^ do begin
                    inc:=inc +1;
                    lsp:=lsp^.nextlist;
                end;
            end;
        end;
    end;
end;

procedure BegTime;
var i,j:integer; vrt:ppoint; lvrt:list;
    ttec: integer;
begin
    for i:=vert to npnt do
        with pnt[i]^ do
            begin
                lvrt:=last;
                ttec:=tor+t;
                while (lvrt <> nil) do
                    begin

```

```

        vrt:=pnt[lvrt^.ipoint];
        with vrt^ do
            if ttec > tor then tor:=ttec;
            end;
        lvrt:=lvrt^.nextlist;
    end;
end;
tcp:=pnt[npnt]^tor+pnt[npnt]^t;
writeln(f, '    max dir crok=', tcp);
end;
{Данная процедура находит поздние окончания работ}

```

```

procedure LastTime;
var i,j:integer; vrt:ppoint; ttec: integer;
    lvrt:list;
begin
    for i:=0 to npnt do
        pnt[i]^tfin:=tpt;
    for i:=npnt downto 0 do
        with pnt[i]^ do begin
            ttec:=tfin;
            lvrt:=first;
            while (lvrt <> nil) do begin
                vrt:=pnt[lvrt^.ipoint];
                with vrt^ do begin
                    if tfin > ttec -t then
                        tfin:=ttec-t;
                    lvrt:=lvrt^.nextlist;
                end;
            end;
        end;
    for i:=0 to npnt do
        dirsrok[i]:= pnt[i]^tfin;
end;
begin end.

```

```

unit delsmain;
    {основная процедура жадного метода}
interface
uses dmydata,dbeglast,dselect, destimels,dstepfor,dnewrec,doutrasp;

```

```

procedure ElsMain(var a,b:integer);

implementation
procedure ElsMain;
var i,tmin,dlina, estmin:integer;
begin tmin:=0;
      setw(pnt[0],1,k,tmin); k:=1;
      while (k>0) and (k<=npnt) do begin
        min:=10000;
        For i:=1 to s do begin
          if min > time[i] then begin
            min:=time[i];
            tecpr:=i;
          end;
        end;
        tecproc:=tecpr; {номер текущего процессора}
        tmin:=time[tecpr];
        dlina:=mindlina;
        SelectJob(k,tmin,tecjob);
        setw (tecjob,tecproc,k,tmin);
        inc(k);
      end;
      newrecord;
end;
begin end.

{процедура оценки решения}
unit destimels;
interface
uses dmydata;
function cross(x1,x2,y1,y2,l1,t1: Integer): Integer;
implementation
function cross;
var lf, rf, lr, lm, j:integer;
begin cross:= 0;
      lf:= x1 -y1;
      rf:= y2 - x2;
      if (lf >0) and (rf>0) then begin
        lr:=min1(lf,rf); lm:=min1(l1,t1);
        cross:= min1(lr,lm);
      end;
end;

```



```

    end;
end;
begin end.

{процедура оценки решения, работает один раз}
unit destone;

interface
uses dmydata;
function cross(x1,x2,y1,y2,l1,t1: Integer): Integer;

implementation
function cross;
var lf,rf,lr,lm,j:integer;
begin cross:=0;
    lf:= x1 -y1;
    rf:= y2 - x2;
    if (lf >0) and (rf>0) then begin
        lr:=min1(lf,rf); lm:=min1(l1,t1);
        cross:= min1(lr,lm);
    end;
end;
begin end.

unit dinittask;
interface
uses dmydata,dbegels, dbeglast,destone;
procedure initels(var f:text);
{работает один раз для инициализации, нужно вставить правильную нижнюю оценку.}
implementation

procedure initels;
var a1,i:integer;
begin
    BegTime(f,npnt,nedg,0);{для каждой работы считаем ранние окончания}
    s2:=s1; s1:=s1 div s; if s1*s=s2 then dec(s1);
    tcp:=pnt[npnt]^tor+pnt[npnt]^t;
    mindlina:=max(s1+1,tcp); { Это максим. дир срок}
    a:=max(0, s1 + 1- tcp); {нижняя оценка длины расписания}
    a1:=a+1;

```

```

LastTime(f,npnt,nedg,npnt,tcp);
{Здесь выбирается тип алгоритма}
sortdd; {Выбираем задание с минимумом суммы D+d}
{Варианты сортировок - sortdt выб-м зад-е с мин-м D-t}
{sortd - выб-м зад-е с мин-м дир ср}
{sortds - зад-е с мин-м ранним сроком}
b:=s2;
lbound:=a;
writeln(f,'нижняя оценка целевой функции=' , a);
end;
begin
end.

unit dmydata;
interface
const
MaxPnt=102;{здесь устанавливается число заданий для постановки расписание}
{MaxPnt = N+2}
Infty=1E20;
Maxpr=8;{здесь устанавливается число процессоров}
gz=10000;
l1=6;
type
pedge=^tedge;
tedge=record {ребро графа --- начало, конец и время}
    beg,en:integer; {начало и конец ребра}
    tf: integer;{позднее начало работы}
end;
list=^tlist;
tlist=record
    ipoint: integer;
    nextlist: list;
end;
ppoint=^tpoin;
tpoin=record {вершина графа задание}
    first: list; {первая вершина в списке входящих}
    last: list; {первая вершина выходящих}
    inc: integer; {число входящих дуг, число предшественников}
    t: integer; {время выполнения задания}
    ijob: integer; {номер задания}

```

```

    timbeg: integer; {время начала работы в расписании}
    res: integer; {время простоя процессора перед началом выполнения}
    nproc: byte; {номер процессора в расписании}
    tor,tfin: integer; {ранние начала и поздние окончания задания}
end;
tablestr=record{строка в таблице для оценок результата}
    name: string;
    lbb: integer; {нижняя граница}
    fl: integer ;
    optl: Integer;{время выполнения оптимального расписания}
    rlopt: Real;{оценка приближенного реш-я Topt>0 -> tbl.fl/Topt -1 else Topt}
    rllowb: Real;{нижняя оценка lbound>0 -> =fl/lbound-1 else Topt}
end;
ptrrec=^tablestr;
mas=array[1..Maxpr] of integer;
vertmas=array[0..MaxPnt] of ppoint;
masvrt= array[0..MaxPnt] of integer;
reztable=array[0..180] of tablestr;
{описание типов}

var    tCount, tSum: TDateTime; {для подсчета времени работы программы}
    nedg,npnt,m,mindlina: integer;
        {количество вершин и ребер,процессоров}
    pnt: vertmas;          {основной массив для хранения вершин-работ}
    pedg, dirsrok: masvrt; {массив отсортированных по позднему началу вершин}
    plan, bestplan: vertmas; {массив для хранения перестановки}
    time:mas; {время освобождения процессоров}
    g:masvrt;
    itogtable: reztable;{таблица для вывода результата}
    tbl:tablestr;
    x,critjob,tecjob: ppoint; {рабочая переменная --- задание}
    vrt: ppoint;
    a,b,r,                {для бинарного поиска}
    s1,s2,min,mintf, lm,estotm,tjob,opozd,nodop:integer;
    s,j,tecproc, sum,  iternumber, lbound, tecpr: integer;
    iter,zdel: longint;
    k, k1, tcp, Topt, Optdlina:integer;
    f,f1:text;
    lsp:list;
    prost,bestres:masvrt;

```

```

numrt: array[0..3] of integer;
  {массив для количеств rt =0, <5%, 5%< <10%, >10%, соотв-но}

function min1(a1,a2:integer):integer;
function max(a1,a2:integer):integer;

implementation
  function min1;
  begin If a1 > a2 then min1:=a2 else min1:=a1; end;
  function max;
  begin if a1 > a2 then max:=a1 else max:= a2; end;
begin end.

unit dmaintest1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, dmydata,dbeglast, destimels,dbegels,dreadgr1,dinittask, delsmain,
  doutrasp;
procedure mtest1(var f,f1:text);

implementation
procedure mtest1;
{Данная программа реализует жадный алгоритм построения оптимального расписания
для задачи параллельного упорядочивания с директивными сроками и
временами поступления.}
var i, a1,tmin,nomertest : integer;
    st:string;
    {начало основной программы}
begin
  readgraph(f);
  initels(f1);
  ElsInit;
  ElsMain(a,b);
end;
  {конец программы}
begin end.

unit doutrasp;
interface

```

```

uses dmydata;
procedure outtable(var f:text );
procedure outrez(var f:text;st:string;var tbl:tablestr );
implementation

procedure outrez;{Процедура вывода результата}
begin
    tbl.optl:=Topt;
    tbl.fl:=Topt;
    if Topt > 0 then tbl.rlopt:=tbl.fl/Topt-1 else tbl.rlopt := Topt;
    if lbound > 0 then
        tbl.rllowb:=tbl.fl/lbound-1
        {приближенное решение по отн-ю к ниж границе}
    else    tbl.rllowb := Topt;
    if tbl.rllowb = 0 then
        inc(numrt[0])
    else if tbl.rllowb <= 0.05 then
        inc(numrt[1])
        else if tbl.rllowb <= 0.1 then
            inc(numrt[2])
            else inc(numrt[3]);
    tbl.lbb:=lbound;
    writeln(f,'Полученное расписание   Lopt=',Topt);
    writeln(f,'Длина расписания   Topt=',Ooptdlina);
    writeln(f, tbl.lbb,'   ',tbl.fl,'   ');
    writeln(f);
end;

procedure outtable;{Процедура вывода результата}
var i:integer;
begin
    writeln(f,' Итоговая таблица');
    writeln(f, 'Среднее время работы t = ', tSum*24*60*60);
    writeln(f,'   n=',npnt);
    writeln(f,'   m=',s);
    writeln(f, ' Ограничение на число итераций=', iternumber);
    writeln(f, '
        ', 'Имя файла',
        ', 'LB', ' ', 'Zf');
    for i:=0 to 180 do begin
        writeln(f, itogtable[i].name,'   ', itogtable[i].lbb:3,'   ',

```

```

        itogtable[i].fl:3);
    end;
    for i := 0 to 3 do
        write(f, numrt[i], '  ');
    writeln(f)
end;
begin end.

```

```

        { процедура новый рекорд для класс. алг.}
unit dnewrec;
interface
uses dmydata;
procedure newrecord;

implementation
procedure newrecord;
var i,mdlina: Integer; vrt: ppoint;
begin min:=-1000; mdlina:=0;
    for i:=1 to npnt do begin
        vrt:=pnt[i];
        with vrt^ do begin
            if timbeg - dirsrok[i] > min then min:=timbeg - dirsrok[i];
            if timbeg +t > mdlina then mdlina:=timbeg + t;
        end;
        Topt:= min;
        Optdlina:=mdlina;
    end;
        {сохраняем новый рекорд}
    bestplan:=plan;
    bestres:=prost;
end;
begin end.

```

```

unit dreadgr1;
interface
uses dmydata;
procedure readgraph(var f:text);
implementation

procedure readgraph;{инициализация графа}

```

```

var
  i,ik, i1:integer;
  edge:pedge;
  en: integer;
begin
  reset(f);
  read(f,npnt);{считываем количество вершин в графе,
               количество процессоров присваиваем вручную}
  s:=Maxpr; npnt:=npnt+1;
  {цикл по вершинам} s1:=0;
  for i:=0 to npnt do begin
    new(pnt[i]);
    with pnt[i]^ do begin
      first:=nil; {первая вершина в списке входящих}
      last:=nil;  {первая из выходящих}
      tor:=0; tfin:=gz;
      ijob:=i; res:=0; pnt[i]^.ijob:=i;
      nproc:=0; timbeg:=-1;
    end;
  end;
  for i:=0 to npnt do begin
    read(f,i1); read(f,pnt[i]^t); s1:=s1+ pnt[i]^t;
    read(f,j);
    for ik:=1 to j do begin
      read(f,en); {en предшественник для i вершины}
      new(lsp); lsp^.nextlist:=pnt[i]^first; lsp^.ipoint:=en;
      { определяем входящий список!}
      pnt[i]^first:=lsp;
      new(lsp); lsp^.nextlist:= pnt[en]^last;
      pnt[en]^last:=lsp; lsp^.ipoint:=i;
    end;
  end;
  close(f);
end;
begin
end.

unit dselect;
interface
uses dmydata;

```

```
procedure selectjob (k,tmin:integer; var tecj: ppoint);
```

Implementation

```
procedure selectjob;
var sel: boolean;
    i,j,bet,beta: integer;
begin
    tecj:=nil; {есть ли допустимые задания}
    {повторяем цикл, пока не нашли первое подходящее
    при этом задания упорядочены в массиве pedg по tf}
    for i:=0 to npnt do begin
        vrt:=pnt[pedg[i]];
        with vrt^ do begin
            if( g[ijob]=0) then begin
                bet:=tor; beta:=tor-tmin;
                tecj:=vrt;
                if beta > 0 then begin
                    for j:=i+1 to npnt do begin
                        vrt:=pnt[pedg[j]];
                        {ищем задание, кот. можно поставить без простоя, критический путь}
                        with vrt^ do
                            if ( g[ijob]=0) and (max(tor, tmin) +t <= bet)
                                then begin
                                    tecj:=vrt;
                                    exit;
                                end;
                            end;
                    end;
                end;
            end;
        end;
    end;
begin end.

unit dstepfor;
interface
uses dmydata,dbeglast;
```



```

procedure setw(tjob: ppoint; tpr: integer; var k1, tmin:integer);
  {tpr номер процессора}
implementation
  { процедура назначения задания}
procedure setw;
var i, k: Integer;
begin
  k:=k1;
  with tjob^ do begin
    g[ijob]:=k;{назначаем работу}
    nproc:=tpr;{запоминаем номер процессора}
    if tmin < tor then begin
      res:=tor-tmin;{запоминаем простой}
      timbeg:=tor;{время начала}
    end
    else
      timbeg:=tmin;
      {новое время освобождения процессора}
      time[tpr]:=timbeg+t;
      k1:=k1+1;
  end;setw};
begin end.

```