Автоматика и телемеханика, № 12, 2008

Дискретные системы

PACS 07.05.Kf, 02.10.Ox

© 2008 г. А.А. ЛАЗАРЕВ, д-р физ.-мат. наук,
Е.Р. ГАФАРОВ, канд. физ.-мат. наук
(Институт проблем управления им. В.А. Трапезникова РАН, Москва)

К РЕШЕНИЮ ЗАДАЧИ ПОСТРОЕНИЯ РАСПИСАНИЯ ВЫПОЛНЕНИЯ ПРОЕКТА[1]

Рассматривается задача построения расписания проекта с учетом ограничений на ресурсы (*RCPSP*) и ее частные случаи. Проведен сравнительный анализ известных нижних оценок целевой функции – минимизации общего времени выполнения проекта. Выдвинута гипотеза, что для задачи *RCPSP* без прерываний в обслуживании требований оптимальное значение целевой функции не более чем в два раза больше оптимального значения целевой функции соответствующей задачи с прерываниями. Представлены доказательства гипотезы для случаев задачи с параллельными машинами и без отношений предшествования.

1. Введение

В работе рассматриваются задачи построения расписания на быстродействие (минимизация общего времени окончания обслуживания всего множества требований) с ресурсными ограничениями и отношениями предшествования. Данные задачи часто возникают на практике. Например, при строительстве того или иного объекта на разных стадиях строительства необходимо разное количество трудовых ресурсов, строительной техники, материалов и т.п. Между отдельными стадиями строительства существуют отношения предшествования, обусловленные технологией. Требуется составить расписание выполнения работ, при котором не нарушаются отношения предшествования, ресурсные ограничения, и при этом срок окончания строительства был бы минимальным.

В разделе 2 представлена постановка задачи построения расписания проекта с учетом ограничения на ресурсы (*RCPSP*). В третьем разделе приведены результаты анализа известных нижних оценок для задачи *RCPSP* и новая нижняя оценка. Авторами выдвинута гипотеза, что значения целевой функции для задачи *RCPSP* на быстродействие с прерываниями и без прерываний отличаются не более чем в 2 раза. Эскиз доказательства и рассуждения о практическом применении гипотезы представлен в разделе 4. Там же рассматривается частный случай задачи с одним кумулятивным ресурсом и пустым множеством отношений предшествования.

[1] Работа выполнена в рамках гранта поддержки ведущих научных школ (проект НШ-5833.2006.1) и при финансовой поддержке Фонда содействия отечественной науке.

В разделе 5 приведено доказательство выдвинутой гипотезы для частного случая рассматриваемой задачи (Parallel Machine Scheduling – составление расписаний для параллельных машин). В заключительном разделе 6 проведен анализ полученных результатов.

2. Постановка задачи

Дано множество требований $N = \{1, \ldots, n\}$ и K возобновляемых ресурсов. В каждый момент времени t доступно Q_k единиц ресурса k, $k = 1, \ldots, K$. Заданы продолжительности обслуживания $p_i \in \mathbb{Z}^+$ для каждого требования $i = 1, \ldots, n$. Во время обслуживания требования i требуется $q_{ik} \leqslant Q_k$ единиц ресурса $k = 1, \ldots, K$. После завершения обслуживания требования освобожденные ресурсы в полном объеме могут быть мгновенно направлены на выполнение других требований.

Между некоторыми парами требований заданы ограничения предшествования: $i \to j$ означает, что обслуживание требования j начинается не раньше окончания обслуживания требования i.

Обслуживание требований начинается в момент времени $t = 0$. Прерывания при обслуживании требований запрещены.

Необходимо определить моменты времени начала обслуживания требований S_i, $i = 1, \ldots, n$, так, чтобы минимизировать время выполнения всего проекта $C_{\max} = \max\limits_{i=1,\ldots,n} \{C_i\}$, $C_i = S_i + p_i$. При этом должны быть соблюдены следующие ограничения:

– в каждый момент времени $t \in [0, C_{\max})$ и $\sum\limits_{i=1}^{n} q_{ik}\varphi_i(t) \leqslant Q_k$, $k = 1, \ldots, K$, где $\varphi_i(t) = 1$, если требование i обслуживается в момент времени t и $\varphi_i(t) = 0$ в противном случае, т.е. требования в процессе выполнения должны быть полностью обеспечены ресурсами;

– не нарушаются отношения предшествования, т.е. выполняется $S_i + p_i \leqslant S_j$, если $i \to j$, $i, j \in N$.

Данную задачу будем называть *задачей построения расписания проекта с учетом ограничения на ресурсы* и обозначать *RCPSP* (Resource-Constrained Project Scheduling Problem, как принято называть в англоязычной литературе). За полиномиальное время к данной задаче может быть сведена NP-трудная задача о многомерном ранце.

Регулярно разные исследователи выпускают интегрированные обзоры наиболее значимых результатов по задаче *RCPSP*, например [1]. Для тестирования и сравнения многочисленных алгоритмов решения задачи *RCPSP* создана библиотека тестовых примеров *PSPLIB* [2]. Наиболее "быстрым" на данный момент алгоритмом признан алгоритм ветвей и границ Брукера и др. [3]. Среди метаэвристических алгоритмов стоит выделить алгоритм, описанный в [4]. Экспериментальный анализ некоторых эвристических алгоритмов приведен в [5] и [6]. Точный алгоритм решения задачи *RCPSP* с прерываниями предложен в [7].

Решение задачи *RCPSP* может быть представлено в виде набора моментов начала обслуживания требований $S = (S_1, \ldots, S_n)$. Решение, удовлетворяющее ресурсным ограничениям и ограничениям предшествования, будем называть допустимым.

Можем представить структуру проекта как *требования-в-вершинах* ориентированного графа $G = (V, A)$, где каждой вершине из $V = \{1, \ldots, n\}$ соответствует некоторое требование множества $N = \{1, \ldots, n\}$, а множество дуг $A = \{(i, j) \mid i, j \in \in V : i \to j\}$ соответствует ограничениям предшествования. Очевидно, что допустимое решение существует только когда граф предшествования ацикличен.

Обычно в рассмотрение вводят два фиктивных требования 0 и $n+1$ с продолжительностями обслуживания $p_0 = p_{n+1} = 0$. Отношения предшествования $0 \to j \to n+1$, $j = 1, \ldots, n$, $q_{0k} = q_{n+1k} = 0$, $k = 1, \ldots, K$.

Будем обозначать через UB верхнюю границу для оптимального значения $C_{\max}$. К примеру, можем принять $UB = \sum\limits_{i=1}^{n} p_i$.

Для каждого требования $i \in N$ определим временно́е окно $[r_i, d_i]$, в котором требование i должно быть обслужено при любом допустимом расписании S:

$$r_0 = 0; \quad r_i = \max_{j|(j,i)\in A} \{r_j + p_j\}, \quad i = 1, \ldots, n+1;$$
$$d_{n+1} = UB; \quad d_i = \min_{j|(i,j)\in A} d_j - p_j, \quad i = n, n-1, \ldots, 0.$$

2.1. Алгоритм диспетчеризации для задачи RCPSP

Далее приведем популярный алгоритм построения расписания выполнения проекта [3].

Алгоритм 1 (Алгоритм List Scheduling (LS)).

1. Пусть EL – список всех требований без предшественников.

$$Q_k(\tau) = Q_k \quad \forall \tau, \quad k = 1, \ldots, K.$$

2. Если $EL = \emptyset$, то переходим к шагу 10;

3. Выберем требование $j \in EL$;

4. $t := \max\limits_{i|(i,j)\in A} \{S_i + p_i\}$. Если для требования j предшественники не определены, тогда $S_j = 0$;

5. Если существует такой ресурс k, что $q_{jk} > Q_k(\tau)$ для некоторого $\tau \in [t, t+p_j)$, тогда вычислим минимальное значение $t_k > t$, такое, что работа j может быть выполнена на интервале $[t_k, t_k + p_j)$, если рассматривается только ресурс k. Если такого ресурса k нет, то переходим к шагу 7;

6. Положим $t := t_k$. Переходим к шагу 5;

7. Назначим обслуживание требования j на интервал $[S_j, C_j) = [t, t+p_j)$;

8. Резервируем ресурсы под требование $Q_k(\tau) = Q_k(\tau) - q_{jk}$, $k = 1, \ldots, r$, $\tau \in [t, t+p_j)$;

9. $EL = EL \setminus \{j\}$. Добавляем в EL последователей требования j, для которых все предшественники расставлены. Переходим к шагу 2;

10. Конец алгоритма.

Полученное решение зависит от выбора требования j на шаге 3 алгоритма. Идея данного алгоритма заключается в том, что требование j ставится в расписание с наиболее раннего момента времени, при котором не нарушаются ресурсные ограничения и ограничения предшествования. Трудоемкость алгоритма $O(n^2K)$ операций.

Активным назовем такое допустимое расписание (решение) $S = (S_1, \ldots, S_n)$, для которого не существует другого допустимого расписания $S' = (S'_1, \ldots, S'_n)$, такого, что $S'_j \leqslant S_j$ $\forall j \in N$, хотя бы одно из неравенств строгое. Алгоритмом LS строятся только активные расписания. Очевидно, что оптимальное расписание следует искать среди множества активных.

В следующей теореме доказано, что активному расписанию $(S_1, \ldots, S_n)$ соответствует некоторая перестановка элементов множества N, которую будем обозначать через $\pi = (j_1, \ldots, j_n)$.

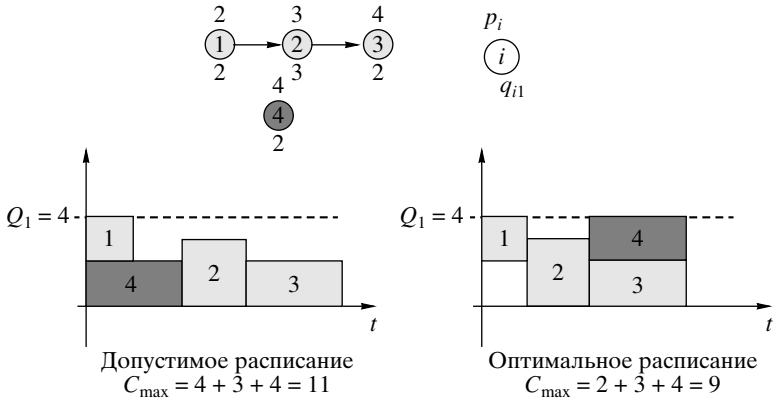

Рис. 1. Пример задачи *RCPSP*.

Теорема 1 [3]. *Активное расписание однозначно представляется в виде перестановки* $\pi = (j_1, \ldots, j_n)$ *из* n *требований.*

Перестановка π задает последовательность выбора требования j на шаге 3 алгоритма.

Подобный алгоритм зачастую называют "конкурсом", так как требование на шаге 3 может выбираться согласно некоторому "приоритету".

Пример 1.

Рассмотрим пример, представленный на рис. 1. Проект состоит из 4-х требований. На сетевом графике сверху над узлами (для i-го требования) указана продолжительность обслуживания требований p_i, снизу – необходимое для обслуживания требования количество q_{i1} возобновляемого ресурса 1. Всего доступно 4 единицы ресурса 1 (к примеру, доступно 4 монтажника).

На рис. 1 представлено 2 допустимых расписания, при которых соблюдены отношения предшествования, ограничения на ресурсы и продолжительность обслуживания требований.

Продемонстрируем работу алгоритма на примере.

Пример 1. Шаг 0. $EL = \{1, 4\}$;
Шаги 3–9. Пусть $j = 1$. Самое раннее время обслуживания требования 1: $[0, 2)$. $EL = \{4, 2\}$;
Шаги 3–9. Пусть $j = 4$. Самое раннее время обслуживания требования 4: $[0, 4)$. $EL = \{2\}$;
Шаги 3–9. $j = 2$. Самое раннее время обслуживания требования 3: $[4, 7)$, так как на интервале $[2, 4)$ для требования 2 уже не хватает ресурсов (зарезервированы под требование 4). $EL = \{3\}$;
Шаги 3–9. $j = 3$. Самое раннее время выполнения требования 3: $[7, 11)$, так как обслуживание требования 2 заканчивается в момент времени 7. $EL = \emptyset$.

В итоге получаем расписание, при котором время выполнения проекта $C_{\max} = 11$. Можно представить последовательность постановки требований в расписание как перестановку $\pi = (1, 4, 2, 3)$.

Если бы поставили требования в порядке $(1, 2, 4, 3)$ или $(1, 2, 3, 4)$, то получили бы расписание, при котором $C_{\max} = 9$, т.е. расписание, соответствующее перестановке $\pi = (1, 2, 4, 3)$, лучше с точки зрения целевой функции.

На основании данного алгоритма диспетчеризации можно построить точный алгоритм ветвей и границ. Ветвление в нем происходит при выборе требования j.

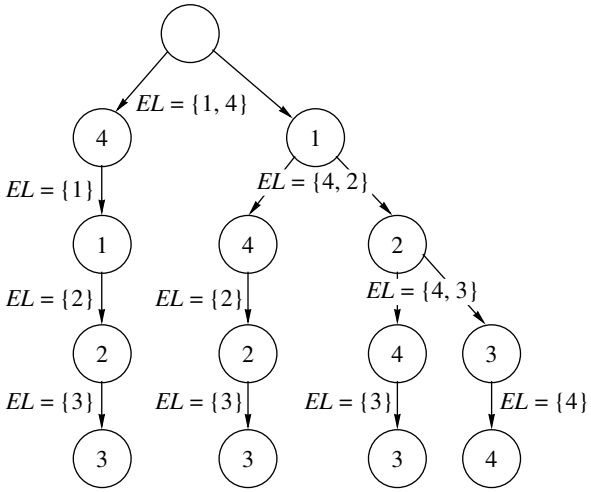

Рис. 2. Дерево поиска оптимального расписания.

Схема ветвления данного алгоритма для примера рис. 1 представлена на рис. 2. Так как алгоритм ветвей и границ является наиболее эффективным для решения данной задачи, то становится актуальным нахождение "хороших" нижних оценок.

2.2. Задача RCPSP с прерываниями обслуживания требований

Задачу *RCPSP*, где разрешены прерывания в обслуживании требований, будем называть *PRCPSP* (Preemption Resource-Constrained Project Scheduling Problem – задача построения расписания проекта с учетом ограничений на ресурсы, с возможностью прерывать обслуживание требований).

Решение задачи *PRCPSP* может быть представлено в виде набора множеств, состоящих из пар чисел $S' = \{S_1 = \{[s_{11}, c_{11}), \ldots, [s_{1m_1}, c_{1m_1})\},\ S_2 = \{[s_{21}, c_{21}), \ldots$ $\ldots, [s_{2m_2}, c_{2m_2})\}, \ldots\}$. Требование 1 обслуживается непрерывно на промежутках $[s_{11}, c_{11}), \ldots, [s_{1m_1}, c_{1m_1})$ с $m_1 - 1$ прерываниями.

Пусть $C_{\max}(S^*)$ – оптимальное значение целевой функции для примера *RCPSP*. Обозначим $C_{\max}(S^*_{pmtn})$ – оптимальное значение целевой функции для того же примера, когда разрешены прерывания.

На рис. 3 представлен пример *RCPSP* и оптимальные решения в случае с прерываниями и без прерываний, когда обслуживается n требований и в каждый момент времени доступно две единицы ресурса.

Сетевой график проекта для данного примера состоит из двух несвязных фрагментов. Первый фрагмент – последовательность "длинных" и "коротких" требований (продолжительностью p и ε соответственно, где ε – достаточно малая величина). Второй фрагмент состоит из одного требования, продолжительностью $(n/2)p$. Значение целевой функции:

$$C_{\max}(S^*) = \left(\frac{n}{2} - 1\right)(p + \varepsilon) + \frac{n}{2}p, \quad C_{\max}(S^*_{pmtn}) = \left(\frac{n}{2} - 1\right)(p + \varepsilon) + p,$$
$$C_{\max}(S^*) - C_{\max}(S^*_{pmtn}) = \left(\frac{n}{2} - 1\right)p < C_{\max}(S^*_{pmtn}),$$
$$C_{\max}(S^*) < 2 \cdot C_{\max}(S^*_{pmtn}),$$

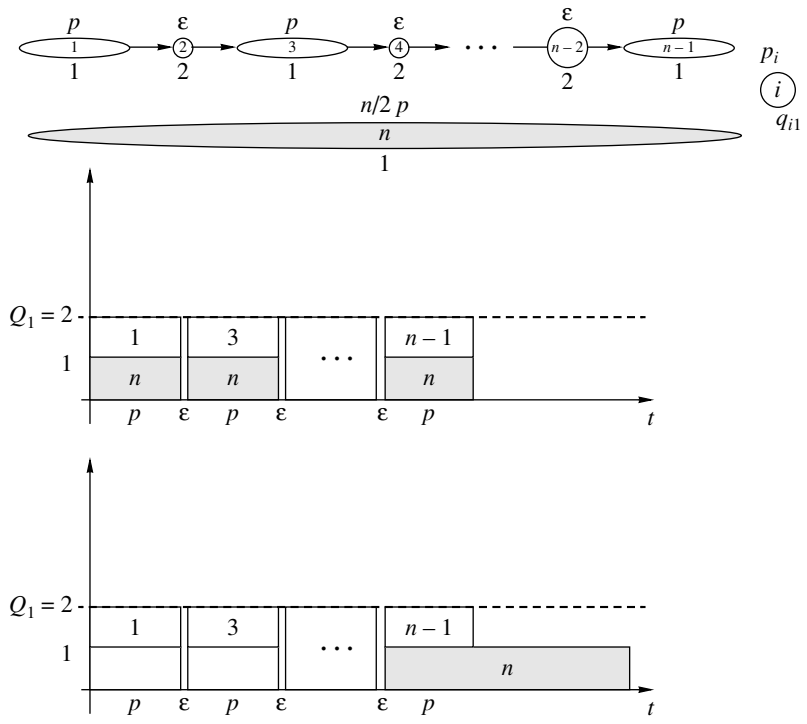

Рис. 3. Пример *RCPSP* с прерываниями.

т.е. время выполнения проекта без прерываний превышает время выполнения проекта с прерываниями не более чем в 2 раза. Оценка асимптотически достигается при $n \to \infty$.

Обратим внимание, что при расписании S^*_{pmtn} каждый фрагмент $l = [s_{nl}, c_{nl})$ требования n обслуживается параллельно с некоторым другим требованием.

3. Относительная погрешность нижних оценок для задачи RCPSP

В данном разделе приводятся результаты анализа нижних оценок для задачи *RCPSP*, в частности относительная погрешность известных нижних оценок.

3.1. LB_0 – длина критического пути

Определение 1. Путь, соединяющий вершины 0 *и* $n+1$ *в сетевом графике и имеющий наибольшую длину, называется критическим путем. Его длина складывается из продолжительности обслуживания требований, входящих в этот путь.*

Длина критического пути будет равна $C_{\max}(S^*)$, когда нет ограничений на ресурсы, т.е. длина критического пути является нижней оценкой для $C_{\max}(S^*)$. Эту нижнюю оценку принято обозначать LB_0.

Утверждение 1. Существует пример RCPSP, для которого $\dfrac{C_{\max}(S^*)}{LB_0} = n.$

Доказательство. Рассмотрим пример *RCPSP*, где требования $i = 1, \ldots, n$ имеют продолжительности обслуживания $p_i = p$. Ограничений предшествования

между требованиями нет. Требуется всего один ресурс, причем $q_{i1} = Q_1$, $i = 1, \ldots, n$. Очевидно, что $LB_0 = p$. Но $C_{\max}(S^*) = pn$, так как никакие два требования не могут обслуживаться параллельно в силу ограничения на ресурсы $q_{i1} = Q_1$, $i = 1, \ldots, n$. Тогда $\dfrac{C_{\max}(S^*)}{LB_0} = n$.

Так как граф предшествования ациклический, то оценка LB_0 может быть найдена за $O(n^2)$ операций.

3.2. LB_1 – максимальная загрузка ресурса

Другая нижняя оценка LB_1 может быть найдена за $O(nK)$ операций при рассмотрении каждого ресурса отдельно.

Определение 2. Назовем величину $\sum\limits_{i=1}^{n} q_{ik}p_i$ *суммарной загрузкой ресурса* k.

Очевидно, что $\sum\limits_{i=1}^{n} q_{ik}p_i \leqslant Q_k \cdot C_{\max}(S^*)$, $k = 1, \ldots, K$. Тогда значение

$$LB_1 = \max_{k=1}^{K} \left\lceil \sum_{i=1}^{n} q_{ik}p_i/Q_k \right\rceil$$

является нижней оценкой для $C_{\max}(S^*)$.

Утверждение 2. Существует пример RCPSP, для которого $C_{\max}(S^*) - LB_1 = \sum\limits_{i=1}^{n} p_i - 1$.

Доказательство. Рассмотрим пример *RCPSP*, где требования $i = 1, \ldots, n$ имеют продолжительности обслуживания p_i. Заданы отношения предшествования $i \to i+1$, $i = 1, \ldots, n-1$; $q_{i1} = \varepsilon$, $i = 1, \ldots, n$. Пусть $Q_1 = \sum\limits_{i=1}^{n} p_i\varepsilon$. Очевидно, что $LB_1 = \sum\limits_{i=1}^{n} q_{i1}p_i/Q_1 = 1$. Но $C_{\max}(S^*) = \sum\limits_{i=1}^{n} p_i$. Тогда $C_{\max}(S^*) - LB_1 = \sum\limits_{i=1}^{n} p_i - 1$. Утверждение доказано.

Очевидно, что для примера *RCPSP*, для которого ограничения на ресурсы не заданы, выполняется $C_{\max}(S^*) - LB_1 = \sum\limits_{i=1}^{n} p_i$.

3.3. LB_S – дополнение критического пути

Обозначим множество требований, принадлежащих критическому пути, через CP. Для каждого требования $i \notin CP$ обозначим через e_i максимальную длину интервала, входящего в $[r_i, d_i]$, в котором требование i может обслуживаться параллельно с требованиями критического пути без нарушения ограничений на ресурсы. Если выполняется $e_i < p_i$, то не существует допустимого расписания, при котором $C_{\max}(S^*) = LB_0$. Тогда значение

$$LB_S = LB_0 + \max_{i \notin CP}\{\max\{p_i - e_i, 0\}\}$$

также является нижней оценкой для $C_{\max}(S^*)$.

LB_S может быть найдена за $O(nK|CP|)$ операций, где $|CP|$ – количество требований в критическом пути.

Утверждение 3. Существует пример RCPSP, для которого $\dfrac{C_{\max}(S^*)}{LB_S} = n/2$.

Доказательство. Рассмотрим пример *RCPSP*, где требования $i = 1, \ldots, n$ имеют продолжительности обслуживания $p_i = p$. Ограничений предшествования между требованиями нет. Пусть $q_{i1} = Q_1$, $i = 1, \ldots, n$, тогда $LB_0 = p$. Нетрудно вычислить $LB_S = p + \max\{p - 0, 0\} = 2p$.

Но $C_{\max}(S^*) = pn$, так как никакие два требования не могут обслуживаться параллельно в силу ограничения на ресурсы $q_{i1} = Q_1$, $i = 1, \ldots, n$. Поэтому $\dfrac{C_{\max}(S^*)}{LB_S} = n/2$. Утверждение доказано.

3.4. Нижняя оценка Мингоцци (Mingozzi)

В [8] представлена формулировка задачи *RCPSP* как задачи линейного программирования. Нижняя оценка Мингоцци получена частичной релаксацией исходной модели. В более ранней работе [9] был рассмотрен подобный метод получения нижней оценки. Обобщение данного метода приведено в [10].

Определение 3. Если существует путь из вершины i в вершину j в ориентированном графе, то будем говорить, что между требованиями i и j существуют косвенные *отношения предшествования.*

Если $i \to j$, то будем говорить, что между данными требованиями заданы *прямые* отношения предшествования.

Определение 4. Множество требований $X \subset N$ будем называть допустимым множеством*, если между каждой парой требований $i, j \in X$ нет отношений предшествования, прямых или косвенных, и ограничения на ресурсы не нарушаются,* $\sum\limits_{i \in X} q_{ik} \leqslant Q_k$, $k = 1, \ldots, K$.

Определение 5. Допустимое множество X назовем доминирующим*, если не существует другого допустимого множества Y такого, что выполняется $X \subset Y$.*

Рассмотрим список всех доминирующих множеств $X_1, \ldots, X_f$ и соответствующие им векторы $a^j \in \{0,1\}^n$, $j = 1, \ldots, f$: $a_i^j = 1$, если $i \in X_j$, иначе $a_i^j = 0$, $i = 1, \ldots, n$.

Определим через x_j длину интервала, в котором все требования множества X_j обслуживаются параллельно. Тогда следующая задача линейного программирования позволяет вычислить нижнюю оценку LB_M для $C_{\max}(S^*)$ [8]:

$$(1) \qquad \begin{cases} \min \sum\limits_{j=1}^{f} x_j, \\ \sum\limits_{j=1}^{f} a_i^j x_j \geqslant p_i, \quad i = 1, \ldots, n, \\ x_j \geqslant 0, \quad j = 1, \ldots, f. \end{cases}$$

В данной постановке допускается, что требование i может обслуживаться больше чем за p_i единиц времени, но не меньше. В модели частично нарушаются отношения предшествования и допускаются прерывания обслуживания требований.

В [3] отмечается, что f растет экспоненциально с ростом n. При $n = 60$ имеем $f \approx 300000$, для $n = 90$ даже $f = 8000000$. В [3] приводится также эффективная техника вычисления нижней оценки LB_M. Но результаты экспериментов показали, что "качество оценки плохое". Тем не менее оценка является одной из самых "сильных" на данный момент.

Утверждение 4. Вычисление оценки LB_M является NP-трудной задачей.

Доказательство. Рассмотрим NP-трудную задачу упаковки в паллеты. Даны стандартные паллеты длиной W и n предметов, каждый длиной w_i, $i = 1, \ldots, n$. Необходимо упаковать все предметы в паллеты так, чтобы число использованных паллет было минимальным.

Преобразуем исходную задачу упаковки в паллеты в задачу *RCPSP*. Каждое требование $i = 1, \ldots, n$ соответствует предмету i. Зададим $p_i = 1$, $q_{i1} = w_i$, $i = 1, \ldots, n$, $Q_1 = W$. Отношения предшествования между требованиями не заданы. Тогда $C_{\max}(S^*)$ соответствует минимальному необходимому количеству паллет в исходной задаче.

Очевидно, что $LB_M = C_{\max}(S^*)$ для построенного примера *RCPSP*. Следовательно, задача нахождения оценки LB_M эквивалентна решению NP-трудной задачи упаковки в паллеты. Утверждение доказано.

В результате вычисления оценки LB_M обслуживание некоторых требований может прерываться. Тогда верно следующее утверждение.

Утверждение 5. Существует пример RCPSP, для которого $\dfrac{C_{\max}(S^*)}{LB_M} \approx 2$.

Доказательство. Рассмотрим пример, изображенный на рис. 3. Для данного примера при вычислении LB_M имеем доминирующие множества: $X_1 = \{1, n\}$, $X_2 = \{2\}$, $X_3 = \{3, n\}$, $X_4 = \{4\}, \ldots, X_{n-1} = \{n-1, n\}$ и оптимальное решение соответствующей задачи ЛП (1): $x_1 = x_3 = \ldots = x_{n-1} = p$, $x_2 = x_4 = \ldots = x_{n-2} = \varepsilon$.

Можно так подобрать значения n, p, ε, что получим $\dfrac{C_{\max}(S^*)}{LB_M} \approx 2$. Утверждение доказано.

В результате вычисления оценки LB_M могут быть нарушены некоторые отношения предшествования. Тогда верно следующее утверждение.

Утверждение 6. Существует пример RCPSP, для которого $\dfrac{C_{\max}(S^*)}{LB_M} = 1{,}5$, *"в оценке" нарушены отношения предшествования.*

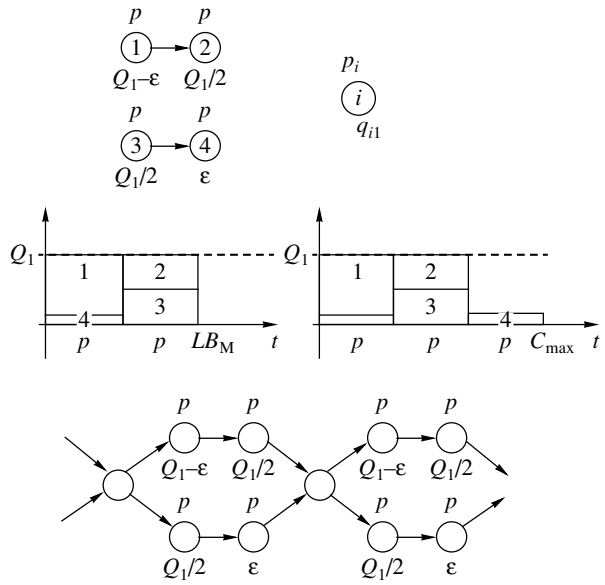

Рис. 4. Рисунок к доказательству LB_M.

Доказательство. Рассмотрим пример, изображенный на рис. 4, когда $Q_1/2 \gg \varepsilon$. При вычислении оценки LB_M получим $X_1 = \{1, 4\}$, $X_2 = \{2, 3\}$, $x_1 = p$, $x_2 = p$, $LB_M = 2p$. Но $C_{\max}(S^*) = 3p$. Тогда $\dfrac{C_{\max}(S^*)}{LB_M} = 1{,}5$. Утверждение доказано.

В [3] представлена модифицированная оценка Мингоцци LB_B, при вычислении которой учитываются $[r_i, d_i]$, $i = 1, \ldots, n$. Для этой оценки утверждения и доказательства аналогичны.

3.5. Оценка LB_{LG}

Усилим оценку LB_1, добавив в рассмотрение $[r_i, d_i]$, $i = 1, \ldots, n$, и *максимально возможный уровень загрузки* в каждой точке t.

Для нахождения оценки LB_{LG} используются алгоритмы решения задачи Разбиение.

Задача 1. **Задача Разбиение (Partition).** *Задано упорядоченное множество из n положительных целых чисел $B = \{b_1, b_2, \ldots, b_n\}$. Требуется разбить множество B на два подмножества B_1 и B_2, $B_1 \bigcap B_2 = \emptyset$ и $B_1 \bigcup B_2 = B$ так, чтобы минимизировать значение:*

$$\left| \sum_{b_i \in B_1} b_i - \sum_{b_i \in B_2} b_i \right| \longrightarrow \min .$$

Опишем **модифицированную задачу Разбиение**, которая потребуется для вычисления LB_{LG}.

Задача 2. Задано упорядоченное множество из n положительных целых чисел $B = \{b_1, b_2, \ldots, b_n\}$ и число $A \leqslant \sum\limits_{b_i \in B} b_i$. Требуется выделить подмножество чисел $B_1 \in B$ так, чтобы минимизировать значение:

$$\left| A - \sum_{b_i \in B_1} b_i \right| \longrightarrow \min .$$

В следующем утверждении показано, что модифицированная задача Разбиение сводится к задаче Разбиение.

Утверждение 7. Если для модифицированной задачи Разбиение существует $B_1 \in B$, что $\sum\limits_{b_i \in B_1} b_i = A$, то для задачи Разбиение с множеством чисел $\overline{B} = B \cup \{b_{n+1}, b_{n+2}\}$, где $b_{n+1} = A + \sum\limits_{b_i \in B} b_i$ и $b_{n+2} = 2 \sum\limits_{b_i \in B} b_i - A$, существуют два подмножества B_1 и B_2, $B_1 \bigcap B_2 = \emptyset$ и $B_1 \bigcup B_2 = \overline{B}$, $\sum\limits_{b_i \in B_1} b_i = \sum\limits_{b_i \in B_2} b_i$.

Доказательство. Сведем модифицированную задачу к исходной. Дополним множество B двумя требованиями: $b_{n+1} = A + \sum\limits_{b_i \in B} b_i$ и $b_{n+2} = 2 \sum\limits_{b_i \in B} b_i - A$ и рассмотрим задачу Разбиение с множеством чисел $B \cup \{b_{n+1}, b_{n+2}\}$.

Очевидно, что числа b_{n+1} и b_{n+2} окажутся в разных подмножествах B_1 и B_2. Пусть выделим подмножество чисел $B_1 \in B$ так, что $A = \sum\limits_{b_i \in B_1} b_i$. Рассмотрим два множества

$$\overline{B_1} = \{b_{n+2}\} \cup B_1 \quad \text{и} \quad \overline{B_2} = \{b_{n+1}\} \cup B \setminus B_1.$$

Тогда

$$\sum_{b_i \in \overline{B_1}} b_i = A + 2\sum_{b_i \in B} b_i - A = 2\sum_{b_i \in B} b_i$$

и

$$\sum_{b_i \in \overline{B_2}} b_i = A + \sum_{b_i \in B} b_i + \sum_{b_i \in B} b_i - A = 2\sum_{b_i \in B} b_i,$$

т.е. $\sum\limits_{b_i \in \overline{B_1}} b_i = \sum\limits_{b_i \in \overline{B_2}} b_i$. Утверждение доказано.

Очевидно, что задачу Разбиение можно свести к модифицированной задаче Разбиение, приняв $A = \dfrac{1}{2}\sum\limits_{b_i \in B} b_i$, т.е. можно говорить об эквивалентности обеих задач.

Для обеих задач Разбиение в [11] приводится эффективный графический алгоритм, который на практике зарекомендовал себя лучше известных алгоритмов динамического программирования [12]. С помощью данного графического алгоритма можно решать примеры с нецелочисленными значениями b_i и A.

Обозначим $z_0 < z_1 < \ldots < z_\tau$ – упорядоченный список различных r_i, d_i значений. Для каждого интервала $I_t = [z_t, z_{t-1})$, $t = 1, \ldots, \tau$, определим множество требований Y_t, для которых $r_i \leqslant z_{t-1} \leqslant z_t \leqslant d_i$.

Для каждого ресурса $k = 1, \ldots, K$ оценка LB_{LG} вычисляется независимо.

Алгоритм вычисления LB_{LG}.

Алгоритм 2.

1. $G := 0$; $LB_{LG} := 0$; $t := 1$;

2. Учитываем объем работ по ресурсу k для требований, которые доступны для обслуживания с момента времени z_{t-1}. $G := G + \sum\limits_{i|z_{t-1}=r_i} q_{ik}p_i$;

3. Построим пример модифицированной задачи Разбиение: $B := \{q_{ik}|i \in Y_t\}$, $A := Q_k$;

4. Решив пример модифицированной задачи Разбиение, найдем подмножество B' и значение $H = \sum\limits_{b_i \in B'} b_i$, $A - \sum\limits_{b_i \in B'} b_i \geqslant 0$. Значение H – *максимально возможный уровень загрузки*;

5. Вычислим длину прямоугольника с высотой H и объемом не больше G. $l := \min\{z_t - z_{t-1}, G/H\}$;

6. $G := G - lH$, $LB_{LG} := z_t + l$.

7. Если $t = \tau$, тогда переходим к шагу 8, иначе $t := t + 1$, и переходим к шагу 2.

8. Конец работы алгоритма.

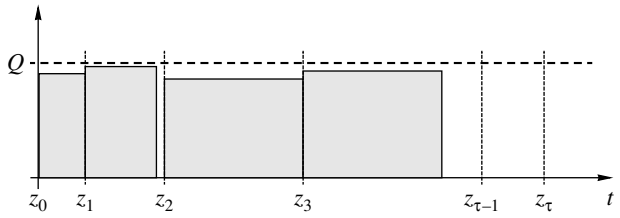

Рис. 5. Оценка LB_{LG}.

Утверждение 8. Найденное в результате работы алгоритма 2 *значение* LB_{LG} *является нижней оценкой для* $C_{\max}(S^*)$.

Доказательство следует из алгоритма вычисления LB_{LG}.

Утверждение 9. Выполняется $LB_{LG} \geqslant LB_0 - p_j$, *где* j *– последнее требование в критическом пути.*

Доказательство. Без ограничения общности примем $LB_0 = d_{\max}$, тогда $LB_0 = d_{\max} = \max\limits_{j \in N} d_j = z_\tau$. При этом $d_{\max} = z_\tau$. По алгоритму имеем $LB_{LG} \geqslant z_{\tau-1} = z_\tau - p_j$, где j – последнее требование в критическом пути. Тогда $LB_{LG} \geqslant LB_0 - p_j$. Утверждение доказано.

В отличие от оценки LB_M данную оценку можно модифицировать, чтобы вычислять нижнюю оценку для задач, когда Q_k зависит от времени t (существуют интервалы времени, на которых $Q_k = Q_k(t)$), что полезно на практике и при реализации некоторых алгоритмов ветвей и границ.

Для случая, когда Q_k постоянно, т.е. не зависит от времени t, оценка LB_{LG} вычисляется не более чем за $O\left(n^2 \sum\limits_{i=1}^{n} q_{ik}\right)$ операций. По результатам экспериментов алгоритмы, представленные в работе [11], позволяют найти решение задачи Разбиение за время $O(n^2)$ для подавляющего большинства примеров. Оценку LB_{LG} можно вычислить за $O(n^3)$ для большинства примеров.

3.6. Оценка LB_{SPP}

Кажется логичным использовать решение задачи *упаковки в полосу* (Strip Packing Problem (SPP)) как нижнюю оценку для задачи *RCPSP*.

Упаковка в полосу.

Задана полоса высотой R и n предметов, которые необходимо упаковать в полосу без пересечений так, чтобы минимизировать задействованную длину полосы. Для каждого предмета $i = 1, \ldots, n$ определена его высота r_i и длина p_i. Переворачивать и разрывать предметы запрещено. Обозначим оптимальную длину полосы через SPP^*.

Каждому примеру SPP "соответствует" пример *RCPSP*, для которого определен один ресурс мощности $Q_1 = R$ и n требований с продолжительностями p_i и потребностями в ресурсе $q_{i1} = r_i$, $i = 1, \ldots, n$. Отношения предшествования между требованиями не заданы.

В [13] показано, что существуют примеры, для которых $SPP^* \geqslant C_{\max}(S^*)$ для соответствующего примера *RCPSP*.

Как было показано выше, все имеющиеся оценки (LB_0, LB_1, LB_s) неэффективны либо их (LB_{LG}, LB_M) нахождение является в общем случае NP-трудной задачей. Отсутствие эффективных методов нахождения нижних оценок не позволяет использовать методы ветвей и границ, branch & cuts и т.п. на практике.

4. Отношение оптимальных значений целевой функции для задач с прерываниями и без прерываний

Лемма 1. Для любого примера PRCPSP существует оптимальное решение, при котором параллельно каждому фрагменту $l = [s_{il}, c_{il})$ *"прерываемого" требования* i *в каждый момент времени* $t \in [s_{il}, c_{il})$ *обслуживаются другие требования, кроме, быть может, последнего фрагмента.*

Доказательство. Предположим, что в оптимальном решении одному из фрагментов l "прерываемого" требования i параллельно не обслуживается ни одно

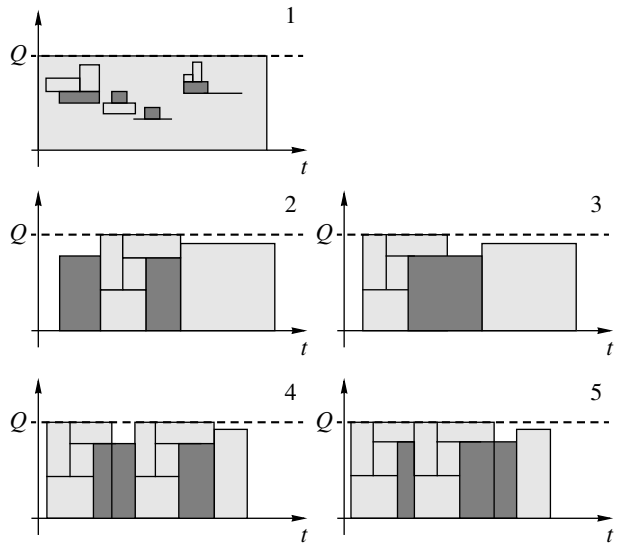

Рис. 6. Рисунок к лемме 1.

требование. Проведем следующее преобразование решения. "Сдвинем" фрагмент l к последующему фрагменту $l+1$ (или к предыдущему фрагменту $l-1$) требования i (см. рис. 6 (2 и 3)). При этом требования, обслуживаемые "между" фрагментами l и $l+1$, сместим "влево" (соответственно "вправо") на величину $c_{il} - s_{il}$. Повторяя такую операцию, получим расписание, при котором требование i не прерывается, причем значение целевой функции не увеличится. В процессе выполнения операции не появляются другие "прерванные" требования.

Возможна ситуация, когда перемещается не весь фрагмент l, а только та его часть, которая "не покрывается" другими требованиями. При это пытаемся переместить эту часть фрагмента к последнему фрагменту m_i (см. рис. 6 (4 и 5)).

Лемма необходима при обосновании следующей гипотезы. Далее будем рассматривать только расписания, удовлетворяющие лемме 1.

Гипотеза 1. *Для любого примера RCPSP и соответствующего примера PRCPSP выполняется*: $C_{\max}(S^*) \leqslant 2 \cdot C_{\max}(S')$.

Авторы выдвигают гипотезу, что оптимальное значение целевой функции (общее время выполнения проекта) в задаче без прерывания обслуживания требований не более чем в два раза превышает значение $C_{\max}$ в соответствующей задаче, когда прерывания разрешены.

Эскиз доказательства. Пусть имеем оптимальное решение S' и значение $C_{\max}(S')$ для примера $PRCPSP$ (с прерываниями). Преобразуем полученное решение S' так, чтобы получить допустимое решение S и значение $C_{\max}(S)$ для соответствующего примера $RCPSP$ (без прерываний). Значение $C_{\max}(S)$ является верхней оценкой для оптимального значения $C_{\max}(S^*)$.

Покажем, что $C_{\max}(S) \leqslant 2 \cdot C_{\max}(S')$. Тогда выполняется $C_{\max}(S^*) \leqslant C_{\max}(S) \leqslant$ $\leqslant 2 \cdot C_{\max}(S')$ и гипотеза будет доказана.

Преобразование решения S' к допустимому решению S проведем следующим образом.

Одно смещение. *Пусть при расписании S' обслуживание только одного требования i прерывается.*

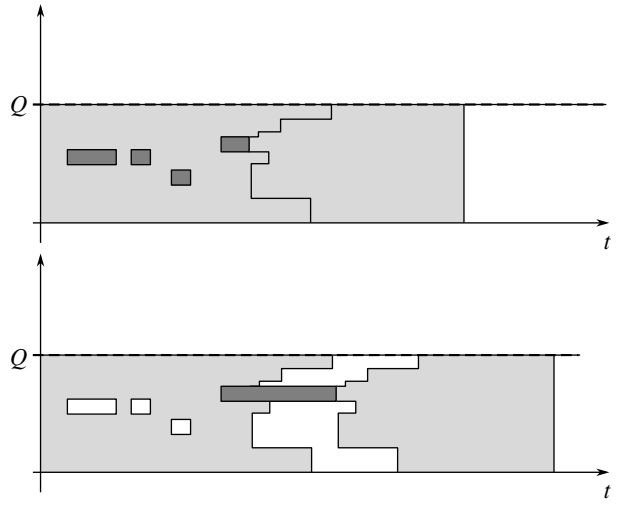

Рис. 7. Одно смещение.

Назначим обслуживание этого требования "целиком" в интервале $[s_{im_i}, s_{im_i} + p_i)$. При этом все требования j, для которых $s_{j1} \leqslant c_{im_i}$, "остаются" на своих местах. Остальные требования смещаются "вправо" на величину равную $p_i - (c_{im_i} - s_{im_i})$ (см. рис. 7).

В результате такой операции значение $C_{\max}$ увеличилось на величину $p_i - (c_{im_i} - s_{im_i})$, т.е. менее чем в 2 раза.

Продолжительность обслуживания "несмещенного" фрагмента превышает p_i *в силу* **леммы 1**.

Аналогично доказывается случай, когда при расписании S' прерывается обслуживание только двух требований (**два смещения**).

Три и более смещений авторы пока не рассматривали.

Доказательство или опровержение исследуемой гипотезы приведут к следующим результатам:

– если гипотеза верна, то, решив задачу с прерываниями, получим нижнюю оценку $LB = C_{\max}(S')$ и недостижимую верхнюю оценку $UB = 2 \cdot C_{\max}(S')$ для задачи *RCPSP* без прерываний. Верхнюю оценку можно использовать в методе ветвей и границ, эффективно отсекая "плохие" решения. Наоборот, получив $C_{\max}(S^*)$ для задачи *RCPSP*, найдем верхнюю оценку для задачи *PRCPSP*;

– если гипотеза окажется неверна и получится $C_{\max}(S^*) \approx O(n) \cdot C_{\max}(S')$ (т.е. разница в порядка n раз), то "лучшая известная" нижняя оценка LB_M, используемая в методе ветвей и границ, может отклоняться от $C_{\max}(S^*)$ в $O(n)$ раз. В этом случае использование метода типа ветвей и границ (branch & bounds, Constraint Programming и т.п.) с нижней оценкой LB_M окажется неэффективным в общем случае.

4.1. Доказательство гипотезы для случая задачи RCPSP с одним ресурсом

Рассматривается частный случай задачи *RCPSP* с одним кумулятивным ресурсом мощности Q_1 и пустым графом отношений предшествования. Этот частный случай является NP-трудным. При $q_{j1} = 1$ и $Q_1 = m$ получаем классическую задачу – m-разбиение [14]. Заметно некоторое сходство данного частного случая с задачей упаковки в полосу (SPP). Тем не менее имеются существенные отличия (см. подраздел 3.6). К примеру, при визуальном представлении задачи *RCPSP* "прямо-

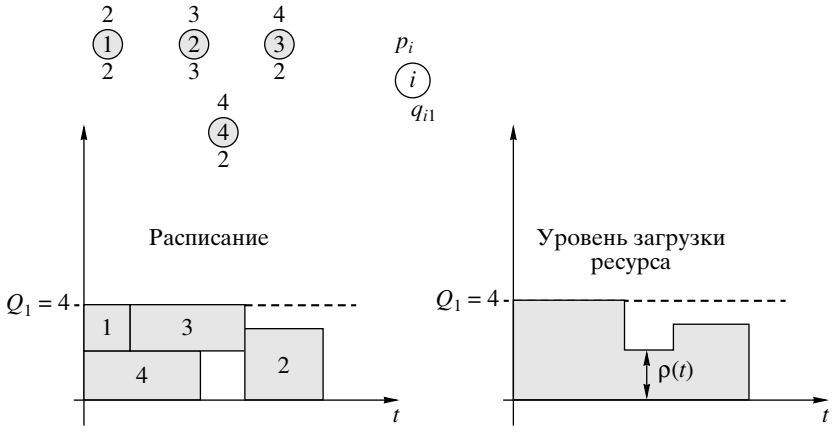

Рис. 8. Пример задачи *RCPSP*.

угольники", соответствующие требованиям, могут быть разорваны по вертикали, что недопустимо в задаче SPP.

Рассмотрим алгоритм диспетчеризации LS со следующим правилом доминирования: из множества EL, еще не поставленных в расписание требований, выбирается то требование, обслуживание которого может начаться раньше других требований из EL без нарушения ресурсных ограничений. Другими словами, на шаге 2 алгоритма LS каждому требованию $j \in EL$ ставится в соответствие "возможное время начала" r_j, с которого требование может обслуживаться без нарушения ресурсных ограничений. Выбирается требование $j \in EL$ с наименьшим значением r_j. Алгоритм диспетчеризации с таким правилом доминирования будем обозначать LS_t.

Пример 2.

Рассмотрим известный авторам пример (только без отношений предшествования), представленный на рис. 8. Продемонстрируем работу алгоритма LS_t на примере.

Шаг 1. $EL = \{1, 2, 3, 4\}$;

Шаги 3–9. $r_1 = r_2 = r_3 = r_4 = 0$. Пусть $j = 1$. Самое раннее время выполнения требования 1: $[0, 2)$. $EL = \{2, 3, 4\}$;

Шаги 3–9. $r_3 = r_4 = 0$, $r_2 = 2$. Пусть $j = 4$. Самое раннее время выполнения требования 4: $[0, 4)$. $EL = \{2, 3\}$;

Шаги 3–9. $r_3 = 0$, $r_2 = 4$. Тогда $j = 3$. Самое раннее время выполнения требования 3: $[2, 6)$. $EL = \{2\}$;

Шаги 3–9. $r_2 = 6$, $j = 2$. Самое раннее время выполнения требования 2: $[6, 9)$. $EL = \emptyset$.

На рис. 8 схематично представлено полученное расписание и соответствующий ему уровень загрузки ресурса.

Теорема 2. $C_{\max}(LS_t) - p_{\max} < 2 \cdot C^*_{\max}$, *где* $p_{\max} = \max\limits_{j \in N} p_j$.

Доказательство. Пусть для некоторого примера I, удовлетворяющего рассматриваемому частному случаю, алгоритмом LS_t построено допустимое расписание $S = (S_1, S_2, \ldots, S_n)$ и найдено значение целевой функции $C_{\max}(LS_t)$.

Рассмотрим уровень загрузки кумулятивного ресурса в каждый момент времени $t \in [0, C_{\max}(LS_t))$. Обозначим $N(t) \subseteq N$ – множество требований, обслуживаемых в момент времени t. Будем называть *уровнем загрузки ресурса* в момент времени t величину $\rho(t) = \sum\limits_{j \in N(t)} q_{j1}$. Рассмотрим значения t, для которых $\rho(t) < Q_1/2$. Покажем,

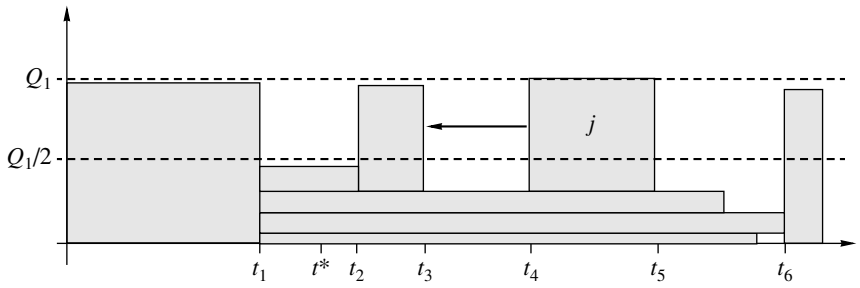

Рис. 9. Рисунок 1 к доказательству теоремы 2.

что на интервале $[0, C_{\max}(LS_t))$ может существовать не более двух таких интервалов $[t_1, t_2)$, $[t_3, t_4)$, $0 \leqslant t_1 < t_2 < t_3 < t_4 \leqslant C_{\max}(LS_t)$, на которых $\rho(t) < Q_1/2$.

Предположим обратное. Пусть существует 3 интервала $[t_1, t_2)$, $[t_3, t_4)$ и $[t_5, t_6)$, $0 \leqslant t_1 < t_2 < t_3 < t_4 < t_5 < t_6 \leqslant C_{\max}(LS_t)$, на которых выполняется $\rho(t) < Q_1/2$. Рассмотрим два случая:

1. Пусть в некоторой точке $t' \in [t_3, t_4)$, уровень загрузки в которой $\rho(t') < Q_1/2$, обслуживается требование $j \in N(t')$. Причем $t_2 \leqslant S_j < t_4$, т.е. требование j не обслуживается ни в одной точке интервала $[t_1, t_2)$. Так как $\rho(t') < Q_1/2$, то $q_{j1} < Q_1/2$, но тогда по алгоритму LS_t требование j должно было начать обслуживаться не позднее момента t_1, так как $\rho(t_1) < Q_1/2$. Получили противоречие.
   Аналогичные рассуждения проводятся для точки $t' \in [t_5, t_6)$ с уровнем загрузки $\rho(t') < Q_1/2$ для требования $j \in N(t')$, где $t_2 \leqslant S_j < t_6$;
2. Пусть не существует требования j, которое обслуживается в некоторой точке $t' \in [t_3, t_4)$, для которого $t_2 \leqslant S_j < t_4$. И не существует требования i, которое обслуживается в некоторой точке $t' \in [t_5, t_6)$ и для которого $t_2 \leqslant S_i < t_6$. Иначе см. п. 1.
   Тогда существует точка $t^* \in [t_1, t_2)$, такая что выполняется $N(t') = N(t^*)$ $\forall t' \in [t_3, t_4)$, и $N(t'') \subseteq N(t^*)$ для некоторой точки $t'' \in [t_5, t_6)$ (см. рис. 9).
   Рассмотрим требование $j \in N(t')$, $t' \in [t_4, t_5)$, $j \notin N(t^*)$. Такое требование существует, так как по нашему допущению $\rho(t') \geqslant Q_1/2$. Но тогда по алгоритму LS_t требование j должно было начать обслуживаться не позднее момента t_3 (см. рис. 9). Получили противоречие.

Итак, авторы показали, что на интервале $[0, C_{\max}(LS_t))$ может существовать не более двух интервалов $[t_1, t_2)$, $[t_3, t_4)$, $0 \leqslant t_1 < t_2 < t_3 < t_4 \leqslant C_{\max}(LS_t)$, на которых выполняется $\rho(t) < Q_1/2$.

Более того, если существует второй интервал $[t_3, t_4)$, то $\forall t \in [t_3, t_4)$ выполняется $N(t) \subseteq N(t^*)$ для некоторой точки $t^* \in [t_1, t_2)$. В дальнейшем интервал $[t_1, t_4)$ будем рассматривать "целиком".

Покажем, что длина такого интервала $t_2 - t_1 \leqslant p_{\max}$. Предположим противное, что $t_2 - t_1 > p_{\max}$.

Тогда существует требование j, $F_j = S_j + p_j = t_2$, для которого $S_j = t_2 - p_j \geqslant$ $\geqslant t_2 - p_{\max} > t_2 - (t_2 - t_1) = t_1$, т.е. $S_j > t_1$. Так как $\rho(S_j) < Q_1/2$, то $q_{j1} < Q_1/2$, но тогда по алгоритму LS_t требование j должно было начать обслуживаться не позднее момента t_1, так как $\rho(t_1) < Q_1/2$. Получили противоречие. Следовательно, длина интервала $t_2 - t_1 \leqslant p_{\max}$ (выполняется и для интервала $[t_1, t_4) : t_4 - t_1 \leqslant p_{\max}$).

Схематично представим полученную загрузку оборудования в виде трех блоков X_1, X_2, X_3 (см. рис. 10). Фрагмент X_1 "находится" в интервале $[0, t_1)$, фрагмент X_2 – в интервале $[t_1, t_2)$ и фрагмент X_3 – в интервале $[t_2, C_{\max}(LS_t))$.

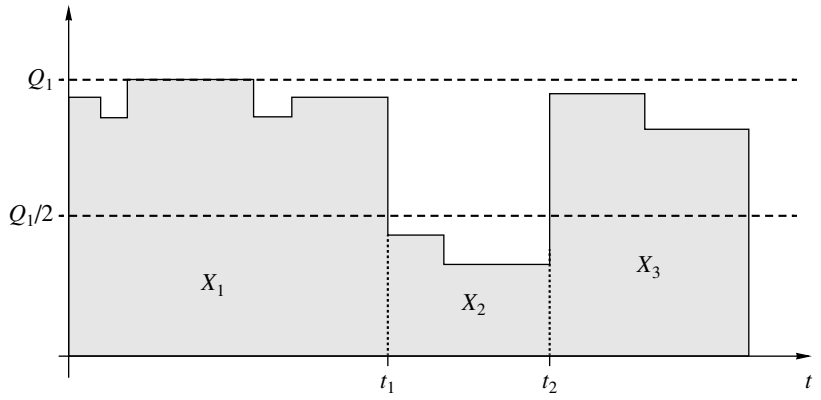

Рис. 10. Рисунок 2 к доказательству теоремы 2.

Суммарную площадь всех трех фигур X_1, X_2 и X_3 будем называть *использованным объемом ресурса* и обозначать через V. Очевидно, что $V = \sum\limits_{j:=1}^{n} p_1 q_{j1}$. Соответственно *неиспользованным объемом ресурса* будем называть величину $V_e = Q_1 \cdot C_{\max}(LS_t) - V$.

Очевидно, полученное алгоритмом LS_t решение можно улучшить с точки зрения целевой функции, если ресурсы использовать "рациональнее", не допуская их простоя, т.е. сократить величину V_e (и соответственно $C_{\max}$).

Рассмотрим пример I', полученный из исходного примера I уменьшением p_j для некоторых требований $j \in N$. Если для некоторого требования $j \in N$ и некоторой точки $t' \in [t_1, t_2)$ выполняется $j \in N(t')$, то примем $p'_j = p_j - (\max(t_1, S_j) - \min(F_j, t_2))$. Для остальных требований положим $p'_j = p_j$.

Рассмотрим расписание $S' = (S'_1, S'_2, \ldots, S'_n)$ для примера I':

$$S'_j = S_j, \quad S_j < t_1;$$
$$S'_j = t_1, \quad S_j \in [t_1, t_2];$$
$$S'_j = S_j - (t_2 - t_1), \quad S_j > t_2,$$

т.е. при расписании S' сокращен фрагмент X_2 (см. рис. 10) и смещен фрагмент X_3 к точке t_1.

Очевидно, что расписание S' допустимо для примера I' и построено алгоритмом LS_t. Обозначим $\overline{C}_{\max}(LS_t)$ значение целевой функции для примера I' при расписании S' и $\overline{C}^*_{\max}$ – оптимальное значение целевой функции для примера I'. Тогда $\overline{C}_{\max}(LS_t) = C_{\max}(LS_t) - (t_2 - t_1)$, $\overline{C}^*_{\max} \leqslant C^*_{\max}$, где $C^*_{\max}$ – оптимальное значение целевой функции для исходного примера I.

При расписании S' $\forall t \in [0, \overline{C}_{\max}(LS_t))$ выполняется $\rho(t) > Q_1/2$, тогда $V > V_e$ для примера I', а следовательно $2 \cdot \overline{C}^*_{\max} > \overline{C}_{\max}(LS_t)$.

Имеем $2 \cdot C^*_{\max} \geqslant 2 \cdot \overline{C}^*_{\max} > \overline{C}_{\max}(LS_t) = C_{\max}(LS_t) - (t_2 - t_1) \geqslant C_{\max}(LS_t) - p_{\max}$.

Обозначим оптимальное значение целевой функции для задачи с прерываниями обслуживания требований через $C^*_{\max}(pmtn)$.

Теорема 3. $2 \cdot C^*_{\max}(pmtn) > C^*_{\max} - p_{\max}$.

Доказательство. Продолжим доказательство предыдущей теоремы. Так как $2 \cdot C^*_{\max}(pmtn) \geqslant 2 \cdot \overline{C}^*_{\max}(pmtn) \geqslant \overline{C}_{\max}(LS_t) \geqslant C_{\max}(LS_t) - p_{\max}$, то отсюда имеем $2 \cdot C^*_{\max}(pmtn) \geqslant C_{\max}(LS_t) - p_{\max} \geqslant C^*_{\max} - p_{\max}$.

5. Задача построения расписания для параллельных машин

В данном разделе рассматривается частный случай задачи *RCPSP* – задача построения расписания для параллельных машин. Дано n требований. Задан граф отношений предшествования и продолжительности обслуживания требований p_j. Требования необходимо обслужить на m параллельных идентичных машинах. Целевая функция $C_{\max}$ – общий момент окончания всех требований. Для данного частного случая имеется единственный кумулятивный ресурс с мощностью $Q_1 = m$, причем $q_{j1} = 1, j = 1, \ldots, n$. Будем обозначать данную задачу PMS (Parallel machine scheduling) или $Pm|prec|C_{\max}$, как принято обозначать в теории расписаний. Данная задача является NP-трудной в сильном смысле [15]. Грэхем [16] показал, что простой алгоритм диспетчеризации дает неплохие результаты. Обозначим $C_{\max}(LS)$ значение целевой функции, полученное алгоритмом диспетчеризации (List Scheduling). Тогда $C_{\max}(LS)/C^*_{\max} \leqslant 2 - \dfrac{1}{m}$, где $C^*_{\max}$ – оптимальное значение целевой функции.

Рассмотрим задачу PMS с прерываниями обслуживания требований $Pm|prec, pmtn|C_{\max}$. Обозначим оптимальное значение целевой функции для этой задачи $C^*_{\max}(pmtn)$.

В пользу выдвинутой авторами **гипотезы 1** говорит следующий факт.

Теорема 4. $C^*_{\max} \leqslant 2 \cdot C^*_{\max}(pmtn)$.

Доказательство. В [16] доказано, что

$$C_{\max}(LS) \leqslant \frac{\sum p_j}{m} + CP,$$

где CP – длина критического пути, т.е. верхняя граница $C_{\max}(LS)$ для задачи $Pm|prec|C_{\max}$ меньше суммы двух простых нижних оценок для этой же задачи.

Нетрудно показать, что $C^*_{\max}(pmtn) \geqslant \dfrac{\sum p_j}{m}$ и $C^*_{\max}(pmtn) \geqslant CP$. Следовательно,

$$C_{\max}(LS) \leqslant 2 \cdot C^*_{\max}(pmtn).$$

Таким образом, для задачи $Pm|prec|C_{\max}$ гипотеза 1 верна. Впервые подобный результат, к сожалению, не опубликовав его, получил Graham R.L. [15].

6. Заключение

В работе исследована задача построения расписания проекта с учетом ограничений на ресурсы и ее частные случаи. Целевая функция – минимизация общего времени выполнения проекта. Проведен сравнительный анализ известных нижних оценок целевой функции. Показано, что вычисление наиболее эффективной нижней оценки Мингоцци является NP-трудной задачей.

Выдвинута гипотеза, что для рассматриваемой задачи без прерываний в обслуживании требований оптимальное значение целевой функции не более чем в два раза больше оптимального значения целевой функции соответствующей задачи с прерываниями. Представлены доказательства гипотезы для случаев задачи с параллельными машинами и без отношений предшествования в обслуживании требований. Доказательство или опровержение исследуемой гипотезы приведут к следующим результатам:

– если гипотеза верна, то, решив задачу с прерываниями, получим нижнюю оценку $LB = C_{\max}(S')$ и недостижимую верхнюю оценку $UB = 2 \cdot C_{\max}(S')$ для задачи *RCPSP* без прерываний. Верхнюю оценку можно использовать в методе ветвей и

границ, эффективно отсекая "плохие" решения. Наоборот, получив $C_{\max}(S^*)$ для задачи $RCPSP$, найдем верхнюю оценку для задачи $PRCPSP$;

– если гипотеза окажется неверна и получится $C_{\max}(S^*) \approx O(n) \cdot C_{\max}(S')$ (т.е. относительная разница значений целевых функций будет порядка n), то "лучшая известная" нижняя оценка LB_M, используемая в методе ветвей и границ, может отклоняться от $C_{\max}(S^*)$ в $O(n)$ раз. В этом случае использование метода типа ветвей и границ (branch & bounds, Constraint Programming и т.п.) с нижней оценкой LB_M окажется неэффективным в общем случае.

СПИСОК ЛИТЕРАТУРЫ

1. *Kolish R., Padman R.* An Integrated Survey of Project Scheduling // Manuscripte aus den Institut fur Betriebswirtschaftslehre. Kiel, Germany, 1997.
2. *Kolish R., Sprecher A.* PSPLIB – A project scheduling problem library // Manuscripte aus den Institut fur Betriebswirtschaftslehre. № 396. Kiel, Germany, 1996.
3. *Brucker P., Knust S.* Complex scheduling. Heidelberg: Springer-Verlag, Berlin, 2006.
4. *Merkel D., Middendorf M., Schmeck H.* Ant Colony Optimization for Resource-Constrainted Project Scheduling // IEEE Trans. Evolut. Comput. 2002. V. 6. № 4. P. 333–346.
5. *Hartmann S., Kolish R.* Experimental evaluation of state-of-the-art heurisitcs for the resource-constrained project scheduling problem // EJOR. 2000. № 127. P. 394–407.
6. *Kolish R., Hartmann S.* Heuristics Algorithms for Solving the Resource-Constrainde Project Scheduling Problem: Classification and Computational Analysis // Manuscripte aus den Institut fur Betriebswirtschaftslehre. № 469. Kiel, Germany, 1998.
7. *Demeulemeester E.L., Herroelen W.S.* An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem // EJOR. 1996. № 90. P. 334–348.
8. *Mingozzi A., Maniezzo V., Ricciardelli S., Bianco L.* An exact alforithm for project scheduling with recource constraints based on new mathematical formulation // Management Sci. 1998. № 44. P. 714–729.
9. *Burkov V.N.* Problems of optimum distribution of resources // Control Cibernet. 1972. V. 1. № 1/2. P. 27–41.
10. Математические основы управления проектами / Под ред. Буркова В.Н. М.: Высш. шк., 2005.
11. *Лазарев А.А.* Графический подход к решению задач комбинаторной оптимизации // АиТ. 2007. № 4. С. 13–23.
12. *Kellerer H., Pferschy U., Pisinger U.* Knapsack Problems. Berlin, Germany: Springer, 2004.
13. *Rykov I.* Approximate solving of $RCPSP$ // Abstract Guide of OR 2006. Karlsruhe 6.09–8.09, Germany. P. 226.
14. *Гэри М., Джонсон Д.* Вычислительные машины и трудно решаемые задачи. М.: Мир, 1982.
15. *Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Schmoys D.B.* Sequencing and Scheduling: Algorithms and Complexity // Report BS-R8909, Centre for Math. Comput. Scie. Amsterdam. 1989.
16. *Graham R.L.* Bounds for certain multiprocessing anomalies // SIAM J. Appl. Math. 1966. № 17. P. 263–269.

Статья представлена к публикации членом редколлегии Д.А. Новиковым.

Поступила в редакцию 04.10.2007