© 2021 г. Б.В. КУПРИЯНОВ, канд. техн. наук (kuprianovb@mail.ru),
А.А. ЛАЗАРЕВ, д-р физ.-мат. наук (jobmath@mail.ru)
(Институт проблем управления им. В.А. Трапезникова РАН, Москва)

ОПТИМИЗАЦИЯ РЕКУРСИВНОГО КОНВЕЙЕРА СВЕДЕНИЕМ К ЗАДАЧЕ УДОВЛЕТВОРЕНИЯ ОГРАНИЧЕНИЙ[1]

Рассматривается задача оптимизации расписания рекурсивного конвейера. Для этого вводится определение конвейера, описываемого связным ациклическим графом, каждая вершина которого представляет собой операцию или функцию управления, ассоциированную с соответствующей рекурсивной функцией из некоторого конечного набора. Каждая рекурсивная функция определяет отношение предшествования операции конвейера. Рассматривается решение задачи минимизации времени выполнения заказа конвейером на конечном множестве возобновляемых ресурсов. Решение осуществляется сведением к задаче удовлетворения ограничений.

Ключевые слова: теория расписаний, балансировка конвейера, flow-shop задачи, задача удовлетворения ограничений.

DOI: 10.31857/S0005231021110052

1. Введение

Задачи RCPSP (Resource-Constrained Project Scheduling Problem) являются одними из основных в теории расписаний. В задачах RCPSP задано множество заказов, которые необходимо обслужить на конечном множестве машин (далее по тексту ресурсов). Заказ тождественен множеству упорядоченных операций и характеризуется временем выполнения. Необходимо минимизировать время выполнения заказов на множестве распределений ресурсов по операциям. Данного вида задачи актуальны и для конвейерных систем [1]. Работы, которые рассматривают методы балансировки сборочного конвейера и планирования ресурсов на этапе проектирования конвейера, приведены в [2]. В [3] приводятся комплексный обзор и анализ различных методов проектирования и планирования конвейерных систем. Большинство работ посвящено балансировке сборочного конвейера (ALB). Для ALB модели предложены методы поиска оптимальных решений с использованием линейного программирования [4, 5] и целочисленного программирования [6]. Из современных работ по оптимизации планирования ресурсов для типовых задач теории расписаний представляют интерес работы [7–10].

Важно отметить две особенности постановки таких задач.

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проект № 20-58-S52006).

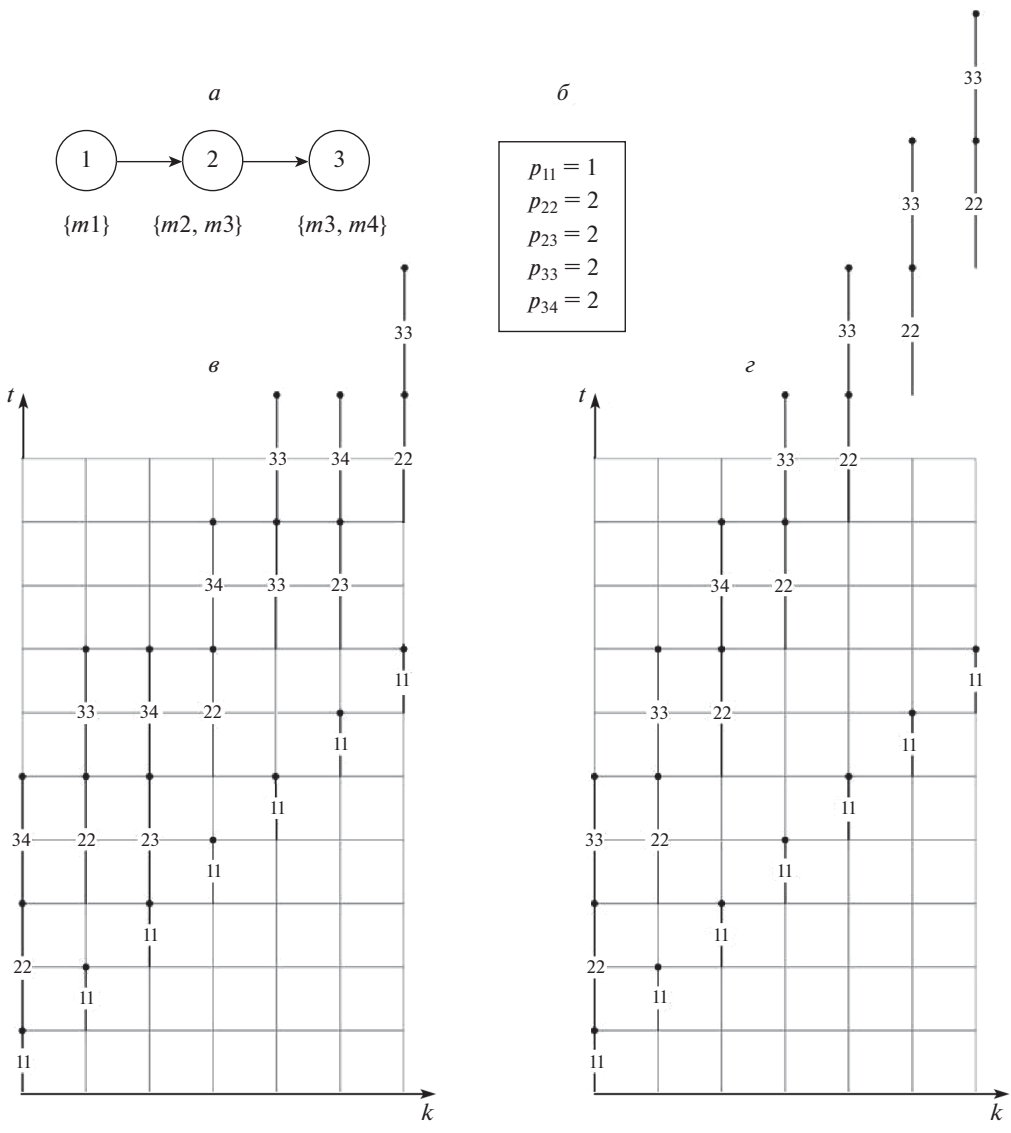

Рис. 1. Пример двух методов распределения ресурсов конвейера.

Первая состоит в том, что ресурсы распределяются до процесса выполнения заказа и на весь процесс выполнения. Это приводит к тому, что каждый ресурс закреплен за какой-либо одной операцией, даже если он может потенциально использоваться несколькими операциями. Однако можно показать на примере, что "перебрасывание ресурса" с одной операции на другую может привести к построению более эффективного расписания. На рис. 1,*а* показана модель конвейера из трех последовательно выполняемых операций. Под операциями в фигурных скобках указаны потенциально используемые ресурсы из множества $\{m1, m2, m3, m4\}$. На рис. 1,*б* указаны длительности операций

при использовании различных ресурсов. Первый индекс — номер операции, второй индекс — номер ресурса. На рис. 1,*в* и 1,*г* показаны временные диаграммы выполнения операций для семи заказов. По оси абсцис номер заказа, по оси ординат время. Отрезок с индексом i, j определяет временной интервал выполнения i-й операции с использованием j-го ресурса для k-го заказа. На рис. 1,*г* показана оптимальная диаграмма с закреплением ресурса за операцией на все время выполнения заказов, а на рис. 1,*в* диаграмма с распределением ресурсов без закрепления за операциями. Из сравнения диаграмм видно преимущество по времени второго метода.

Вторая особенность состоит в использовании ограниченного набора отношений предшествования. Это отношение предшествования между операциями и определяемое с помощью предиката and.

В данной статье предлагается решение, преодолевающее оба эти ограничения. Рассматривается распределение ресурсов с возможностью использования одного ресурса несколькими операциями и с расширением отношений предшествования с помощью рекурсивных функций. Задачи Удовлетворения Ограничений (ЗУО) и методы Constraint Programming [11–13] наряду с прочим используются для решения задач теории расписаний. В данной работе рассматривается решение задачи RCPSP сведением ее к ЗУО применительно к конвейерам, описываемым рекурсивными функциями [14]. Время выполнения операции i зависит от используемого ресурса j, т.е. равно p_{ij}. Примеры описания возможностей и применения таких конвейеров приведены в [15].

Статья имеет следующую структуру. В разделе 2 приведено определение рекурсивного конвейера. Подробно описаны рекурсивные функции. В разделе 3 приведены примеры рекурсивных конвейеров. В разделе 4 описывается определение ЗУО и задача распределения ресурсов конвейера формулируется как ЗУО. Раздел 5 является заключением, в котором рассматривается проблема сложности алгоритма решения поставленной в статье ЗУО.

2. Определение конвейера

Модель и свойства рекурсивного конвейера подробно рассмотрены в [14, 16]. Однако они описаны в предположении, что за каждой операцией закреплен свой ресурс на время выполнения всех заказов. В данном разделе определение рекурсивных функций конвейера дано с учетом распределения ресурса для каждой пары (операция, заказ).

Модель конвейера представляет собой связный ациклический ориентированный граф $G = (V, A)$ с единственной конечной вершиной. V – множество вершин графа и n – количество вершин. A – множество дуг графа, упорядоченных пар вида (v, w), где $v, w \in V$. Вершины графа помечены номерами из множества $I = \{1, \ldots, n\}$ таким образом, что первые n_0 $(1 \leqslant n_0 < n)$ вершин являются начальными, а вершина n конечной. Графическое изображение вершин и дуг графа представлено на рис. 2. Здесь i, j, j_1, j_2 – номера операций (вершин), k – номер заказа, $p_{i,j}$ – время выполнения i-й операции при использовании j-го ресурса и q_i – коэффициент мультиплицирования или ре-

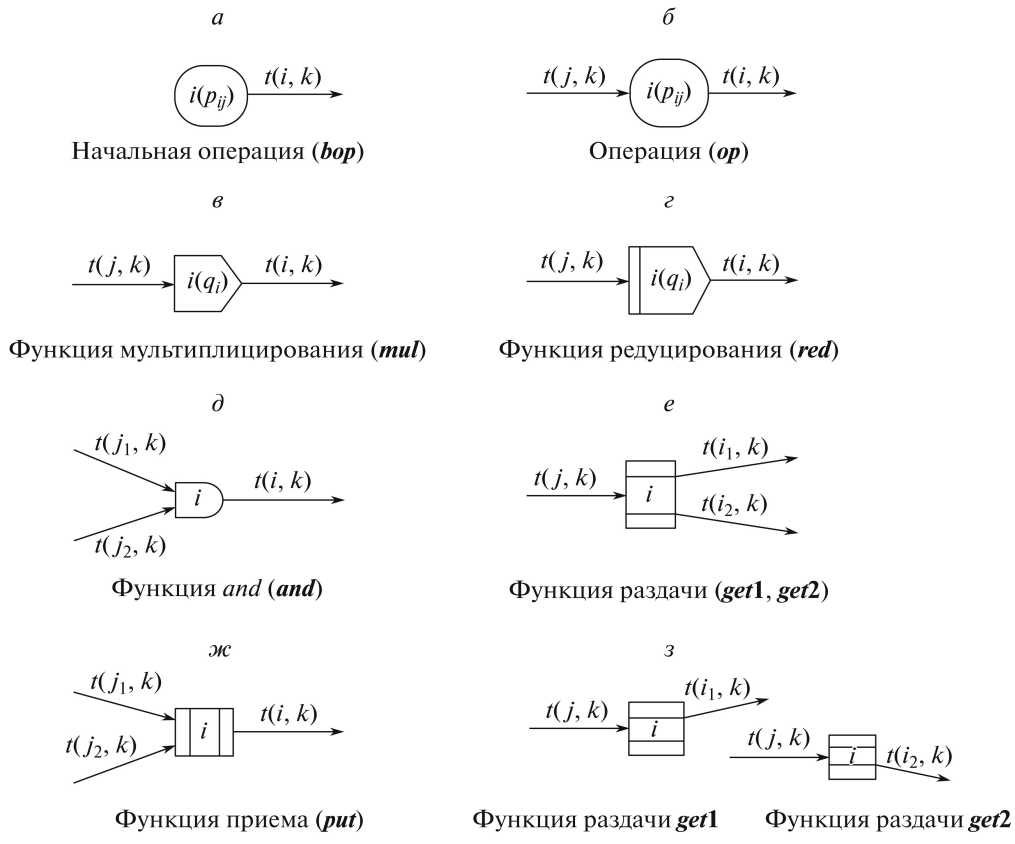

Рис. 2. Графическая нотация операций и спусковых функций конвейера.

дуцирования. Каждая вершина графа может иметь одну входную дугу (см. рис. 2 варианты *б*,*в*,*г*,*е*) или две (см. рис. 2 варианты *д*,*ж*) в зависимости от типа вершины. На множестве номеров вершин графа определены функции предшествования $pred, pred1, pred2 : I \to I$.

Функция $pred(i)$ определена для вершины, имеющей одну предшествующую вершину и вычисляет номер j вершины графа такой, что $(j,i) \in A$.

Следующие две функции определены для вершин, имеющих две предшествующие вершины, условно *вершина* 1 и *вершина* 2:

$pred1(i)$ – вычисляет номер j_1 *вершины* 1 графа такой, что существует дуга $(j_1,i) \in A$;

$pred2(i)$ – вычисляет номер j_2 *вершины* 2 графа такой, что существует дуга $(j_2,i) \in A$.

Каждая вершина графа может соответствовать операции или спусковой функции. Операции ассоциированы с производственными операциями. Спусковые функции определяют отношения временного предшествования операций. Вершины графа помечены с помощью отображения

$$type : I \to E, \quad \text{где} \quad E = \{\mathbf{bop}, \mathbf{op}, \mathbf{and}, \mathbf{mul}, \mathbf{red}, \mathbf{get1}, \mathbf{get2}, \mathbf{put}\}$$

на множество из восьми элементов. Каждая константа множества E обозначает тип вершины графа, а вершина имеет соответствующую графическую нотацию (рис. 2). Конвейер использует множество ресурсов $M = \{m_1, \ldots, m_r\}$. Каждая операция i потенциально может использовать ресурсы из некоторого множества $D_i \subseteq M$. Для каждого $1 \leqslant i \leqslant n$ определим переменную x_{ik} на множестве D_i. Если $x_{ik} = m_j$, то операция i при выполнении k-го заказа использует ресурс $m_j \in D_i$. Время выполнения операции равно p_{ij} и является константой. В каждый момент времени каждый ресурс может использоваться только одной операцией. Прерывания операций запрещены.

Спусковые функции ресурсы не потребляют, и время их выполнения равно 0.

Расписание выполнения операций строится с помощью рекурсивных функций $R : I \times K \times M \to T$, где K-конечное множество номеров заказов, а T-время.

Например, рекурсивная функция

$$t(i,k) = r_1(t(i,k-1), t(j,k), p_{il})$$

или

$$t(i,k) = r_2(t(i,k-1), t(j_1,k), t(j_2,k), p_{il}),$$

где $j = pred(i)$, $j_1 = pred1(i)$, $j_2 = pred2(i)$, вычисляет время завершения обработки i-й операцией k-го заказа при $k \in K$ исходя из времени завершения ею обработки $(k-1)$-го заказа и из времен завершения обработки k-го заказа предшествующими операциями. Рекурсивные функции имеют два аргумента: i – номер операции (номер вершины графа) и k – номер заказа. Для каждого типа вершины, т.е. элемента множества E, своя рекурсивная функция.

Рекурсивная функция $t(i,k)$, вычисляемая по формуле (1), обеспечивает суперпозицию рекурсивных функций, соответствующих вершинам графа, и является обращением к одной из них в соответствии с типом вершины.

$$t(i,k) = \begin{cases} bop(i,k), & \textbf{если} \quad type(i) = \textbf{bop}; \\ op(i,k), & \textbf{если} \quad type(i) = \textbf{op}; \\ and(i,k), & \textbf{если} \quad type(i) = \textbf{and}; \\ mul(i,k), & \textbf{если} \quad type(i) = \textbf{mul}; \\ red(i,k), & \textbf{если} \quad type(i) = \textbf{red}; \\ get1(i,k), & \textbf{если} \quad type(i) = \textbf{get1}; \\ get2(i,k), & \textbf{если} \quad type(i) = \textbf{get2}; \\ put(i,k), & \textbf{если} \quad type(i) = \textbf{put}. \end{cases} \tag{1}$$

Вычисление времени завершения выполнения k-го заказа конвейером осуществляется обращением к рекурсивной функции $t(n,k)$.

Далее описываются все виды вершин: их назначение, тип и соответствующая типу рекурсивная функция.

1. Начальная производственная операция (см. *bop* рис. 2,*а*) с номером $1 \leqslant i \leqslant n_0$ характеризуется временем выполнения k-го заказа p_{ij}, если операция использует ресурс m_j ($x_{ik} = m_j$). Время завершения выполнения данной операцией k-го заказа определяется как время завершения выполнения $(k-1)$-го заказа плюс время выполнения операции p_{ij}. Если операция при выполнении k-го и $(k-1)$-го заказа использует разные ресурсы, то времена их выполнений совмещаются. Время выполнения нулевого заказа равно p_{ij}.

(2) $bop(i,k)$:

$$bop = \begin{cases} p_{ij}, \\ \quad \textbf{если } (x_{ik} = m_j)\&(k = 0); & \text{а)} \\ bop(i,k-1) + p_{ij}, \\ \quad \textbf{если } (x_{ik} = x_{i,k-1})\&(x_{ik} = m_j)\&(k > 0); & \text{б)} \\ bop(i,k-1) + p_{ij} - p_{il}, \\ \quad \textbf{если } (x_{ik} \neq x_{i,k-1})\&(x_{i,k-1} = m_l)\&(x_{ik} = m_j)\&(k > 0). & \text{в)} \end{cases}$$

2. Не начальная производственная операция (см. *op* рис. 2,*б*) с номером $n_0 < i$ характеризуется временем выполнения p_{ij}, если используется ресурс m_j. Время завершения выполнения данной операцией k-го заказа определяется как максимум времен: завершения выполнения данной операцией $(k-1)$-го заказа и завершения выполнения предшествующей ей операцией k-го заказа и плюс время выполнения операции p_{ij}. При этом начала выполнений i-й операцией k-го и $(k-1)$-го заказов совмещаются, если для выполнения используются разные ресурсы. Время выполнения нулевого заказа равно времени выполнения нулевого заказа предшествующей операцией плюс p_{ij}.

(3) $op(i,k)$:

$$op = \begin{cases} t(pred(i),k) + p_{ij}, \quad \textbf{если } (x_{ik} = m_j)\&(k = 0); & \text{а)} \\ t(pred(i),k) + p_{ij}, \\ \quad \textbf{если } (t(pred(i),k) \geqslant t(i,k-1))\&(x_{ik} = m_j)\&(k > 0); & \text{б)} \\ t(pred(i),k) + p_{ij}, \\ \quad \textbf{если } (t(pred(i),k) < t(i,k-1))\& \\ \qquad \&(x_{ik} \neq x_{i,k-1})\&(x_{ik} = m_j)\&(k > 0); & \text{в)} \\ t(i,k-1) + p_{ij}, \\ \quad \textbf{если } (t(pred(i),k) < t(i,k-1))\& \\ \qquad \&(x_{ik} = x_{i,k-1})\&(x_{ik} = m_j)\&(k > 0). & \text{г)} \end{cases}$$

3. Спусковая функция *and* (см. *and* рис. 2,*д*) вычисляет время завершения выполнения k-го заказа для двух предшествующих операций.

(4) $$and(i,k) : \ and = \max(t(pred1(i),k), t(pred2(i),k)).$$

4. Спусковая функция мультиплицирования (см. *mul* рис. 2,*в*) для каждого времени завершения предшествующей ей операции вычисляет q_i времен

завершения операции i. Это означает, что на одно выполнение предшествующей i операции осуществляется q_i ($q_i \geqslant 1$) выполнений операции i.

$$mul(i,k):\ mul = t(pred(i), \lfloor k/q_i \rfloor), \tag{5}$$

где $\lfloor x \rfloor$ обозначает целую часть x.

5. Спусковая функция редуцирования (см. *red* рис. 2,*г*) является обратной по отношению к функции мультиплицирования и вычисляет время завершения выполнения очередной партии из q_i ($q_i \geqslant 1$) заказов.

$$red(i,k):\ red = t(pred(i), (k+1)q_i - 1). \tag{6}$$

6. Спусковая функция раздачи (см. *get* рис. 2,*е*) имитирует разделение конвейерных операций на два потока. Для пользователя она выступает как одна функция, а в реализации она разбивается на две (условно верхнюю и нижнюю) см. рис. 2,*з*. Для верхнего варианта спусковая функция $get1$ вычисляет времена завершений четных заказов, т.е. $t(i,0) = t(p,0)$, $t(i,1) = t(p,2)$, $t(i,2) = t(p,4), \ldots$

$$get1(i,k):\ get1 = t(pred(i), 2k),\ k \geqslant 0. \tag{7}$$

Для нижнего вычисляет времена завершения выполнения нечетных заказов, т.е. $t(j,0) = t(p,1),\ t(j,1) = t(p,3),\ t(j,2) = t(p,5), \ldots$

$$get2(i,k):\ get2 = t(pred(i), 2k+1),\ k \geqslant 0. \tag{8}$$

7. Спусковая функция приема (см. *put* рис. 2,*ж*) является обратной по отношению к функции раздачи и имитирует слияние двух конвейеров в один. Проще всего это пояснить последовательностью значений $t(i,k)$: $t(i,0) = t(p,0),\ t(i,1) = t(q,0),\ t(i,2) = t(p,1),\ t(i,3) = t(q,1),\ t(i,4) = t(p,2),\ t(i,5) = t(q,2), \ldots$

$$put(i,k) : put = \begin{cases} t(pred1(i), k), \\ \qquad \textbf{если } k = 0; \\ \max(put(i,k-1), t(pred2(i), k/2)), \\ \qquad \textbf{если } (k \bmod 2 = 0)\&(k > 0); \\ \max(put(i,k-1), t(pred1(i), (k-1)/2)), \\ \qquad \textbf{если } k \bmod 2 = 1. \end{cases} \tag{9}$$

Здесь $x \bmod y$ – вычисление остатка от деления x на y.

На рис. 3 представлен пример графа конвейера и соответствующей суперпозиции рекурсивных функций для случая, когда каждой операции выделен один ресурс на все время процесса. В реальности такой граф не строится, он является результатом процесса вычисления рекурсивной функции $t(i,k)$.

Время начала функционирования конвейера полагается равным 0.

В предлагаемой модели интерпретация операций и спусковой функции *and* не отличается от таковых в теории расписаний. Остальной набор спусковых функций выработан практикой, на основе построения моделей и является двумя парами прямых и обратных функций ($mul-red$ и $get-put$). Возможно построение новых функций, расширяющих отношение предшествования.

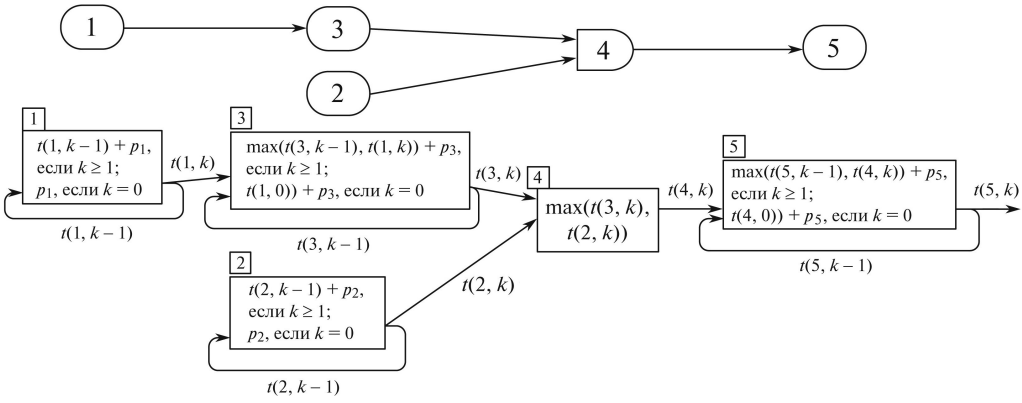

Рис. 3. Пример графа конвейера (сверху) и соответствующего графа суперпозиции рекурсивных функций (снизу).

3. Примеры моделей

Примеры некоторых моделей конвейеров для различных приложений описаны в [15]. В данной статье приведем дополнительные примеры рекурсивных конвейеров. Перед тем как привести примеры, отметим, что временная диаграмма должна составляться для каждой операции отдельно, это ненаглядно

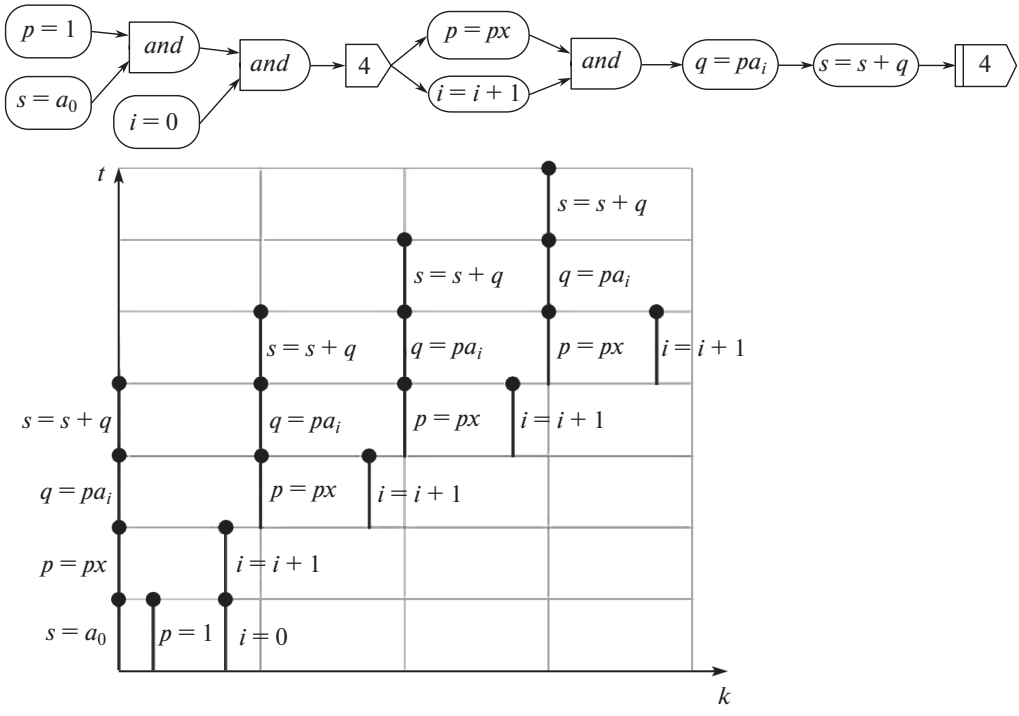

Рис. 4. Пример модели вычислительного конвейера и временной диаграммы.

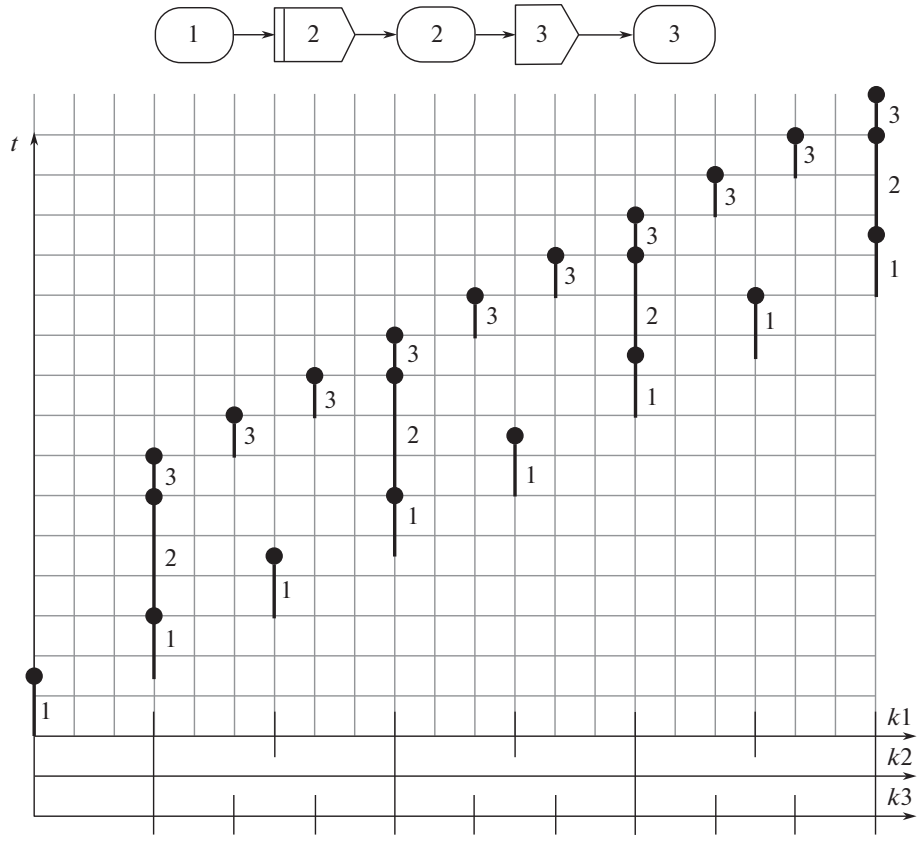

Рис. 5. Пример простого конвейера с функциями мультиплицирования и редуцирования.

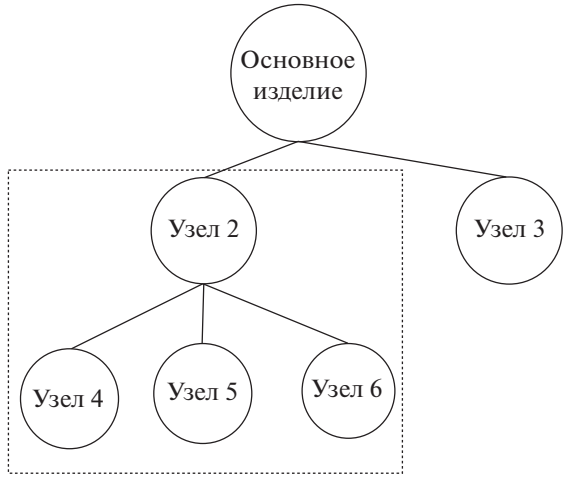

Рис. 6. Структура сборочного изделия.

и громоздко, диаграммы будут совмещены в одной. По оси абсцисс будут отложены номера заказов, а по оси ординат время. Некоторые отрезки времени при таком совмещении в одной диаграмме будут сливаться. Чтобы этого не происходило, отрезки, имеющие по оси абсцис значение k, будут размещаться в интервале между делениями k и $(k+1)$.

Пример 1. На рис. 4 представлена модель конвейера, вычисляющего полином

$$p = a_0 + a_1 x^1 + a_2 x^2 + a_3 x^3 + a_4 x^4,$$

и соответствующая временная диаграмма в предположении, что каждая операция выполняется своим процессором. Времена выполнения всех операций равны 1.

Пример 2. Прежде чем рассмотреть второй пример, разберем вспомогательный. На рис. 5 представлен пример конвейера и его временной диаграммы. Конвейер состоит из трех операций, функции редуцирования с коэффициентом 2 и функции мультиплицирования с коэффициентом 3. На диаграмме видно, что все операции выполняются с разной кратностью, но процесс является конвейерным. Чтобы понять, что такой конвейер может иметь место на практике, рассмотрим пример конвейерной сборки некоторого изделия, которое имеет структуру, представленную на рис. 6. Особенность сборки состоит в том, что узел 2 и узел 3 собираются в отдаленном от основной сборки месте. Поэтому на месте собирается партия из $q2$ узлов, которая потом перевозится на склад основного производства. Аналогично узел 3 собирается в другом отдаленном месте и перевозится партиями по $q3$ узлов. Требуется составить расписание: производства узлов, хранения на складе, перевозки и сборки основного изделия как единого конвейерного процесса. Пример модели такого процесса приведен на рис. 7. Суть ее в том, что узел 2 по мере конвейерной сборки накапливается на складе в количестве $q2$ и потом эта партия доставляется на место основной сборки. Конвейерность этого процесса позволяет описывать функции редуцирования и мультиплицирования. Аналогично для узла 3.

4. Постановка задачи

В данном разделе рассматривается рекурсивный конвейер, представленный в виде ациклического связного графа, для которого необходимо распределением ресурсов минимизировать время выполнения $\hat{k}$ заказов, где $\hat{k}$ – некоторая константа. Данная RCPSP задача сводится к минимизации функции $t(n, \hat{k})$ для заданного $\hat{k}$ на конечном множестве ресурсов M. Опишем формулировку постановки ЗУО для дискретных переменных с конечными множествами значений.

В теории ЗУО [11] представляет собой четверку (V, D, R, C),

где $V = \{x_1, \ldots, x_n\}$ – множество переменных,

$D = \{D_1, \ldots, D_n\}$ – множество доменов переменных,

R – множество отношений различной арности над областями D,

$C = \{C_1, \ldots, C_m\}$ – множество ограничений, связывающих множество значений переменных из V посредством отношений из R.

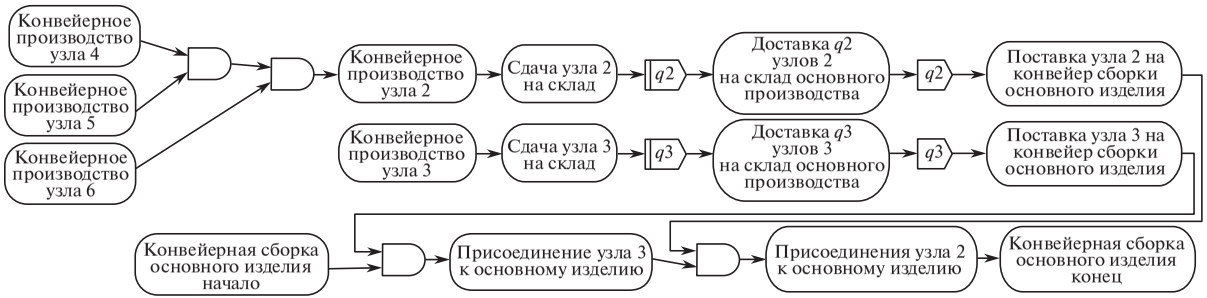

Рис. 7. Модель распределенной конвейерной сборки изделия.

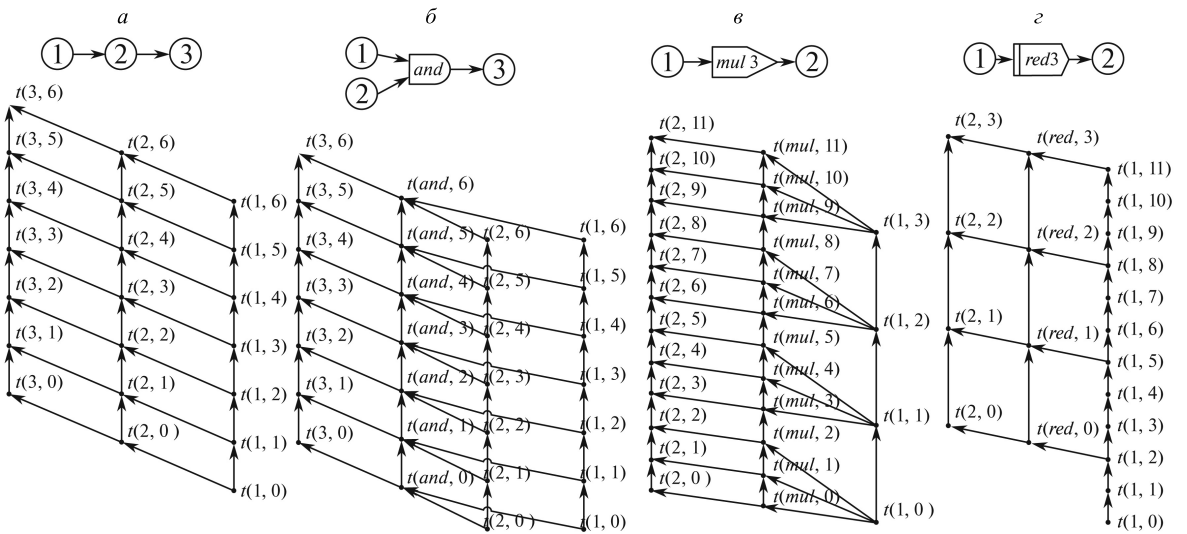

Рис. 8. Примеры графов конвейеров для конкретных значений $\hat{k}$.

Решением ЗУО является нахождение таких значений всех переменных множества V, которые удовлетворяют всем ограничениям. Решений ЗУО может быть одно или несколько.

ЗУО может быть представлена в виде сети ограничений. Было бы естественно в качестве такой сети взять граф модели конвейера, в котором вершине соответствует некоторая операция. Однако в связи с тем, что операция i выполняется k раз при наличии k заказов и в каждом случае может использоваться некоторый ресурс m_i, с операцией связано некоторое множество ресурсов. Чтобы избежать этой ситуации, сгенерируем из исходного графа модели для заданного $\hat{k}$ развернутый граф, в котором каждой вершине соответствует пара (номер операции, номер заказа) и сохраняются отношения предшествования. Построение такого графа возможно для любой модели конвейера и не представляет сложности. Простые примеры таких графов представлены на рис. 8. Сверху представлен исходный граф модели, а ниже граф для конкретного значения $\hat{k}$. Вершины графов помечены соответствующими рекурсивными функциями и ассоциируются с конкретными операциями и заказами. Если исходный граф имеет n вершин (операций), то новый граф имеет $n\hat{k}$ вершин. Легко показать, что развернутый граф так же является ациклическим. Очевидно, что в этом случае каждой вершине соответствует один ресурс, если вершина соответствует операции (тип вершины **bop** или **op**). В остальных случаях, когда вершина соответствует спусковой функции, ресурс не используется. Для отражения этого факта в задаче дополним множество ресурсов нулевым элементом $M = \{m_0, m_1, \ldots, m_r\}$.

Перенумеруем вершины развернутого графа некоторым способом. Пусть $I' = \{1, \ldots, n'\}$, $n' = n\hat{k}$, – множество новых номеров. Будем считать по умолчанию, что $i' \in I'$ обозначает номер вершины нового графа и существуют отображения $\psi : I' \to I$ и $\varphi : I' \to K$. Если в исходном графе существует предшествование по i, то в развернутом графе существует как предшествование по i, так и по k. Определим эти две функции. Функция $pk : I' \to I'$ вычисляет предшествующую по k вершину, т.е. если $i'' = pk(i')$, то $\psi(i') = \psi(i'')$, а $\varphi(i'') = \varphi(i') - 1$. Функция $pi : I' \to I'$ вычисляет предшествующую по i вершину, т.е. если $i'' = pi(i')$, то $\psi(i'') = pred(\psi(i'))$, а $\varphi(i'') = \varphi(i')$. Далее для упрощения записи в выражениях будут использоваться обе нумерации: сквозная со штрихом и двухкоординатная (i, k) без штриха в предположении, что $i' \to (i, k)$, $i = \psi(i')$ и $k = \varphi(i')$.

Перенесем приведенную постановку ЗУО в рассматриваемую область теории расписаний.

Четверка объектов ЗУО будет определена следующим образом:

1) $V = \{x_1, \ldots x_{n'}\}$ – множество дискретных переменных и для каждой из них определено множество значений.

2) $D_{i'} = D_i$ – домен переменной $x_{i'}$, связанный с соответствующей i-й вершиной исходного графа, т.е. $i' \to (i, k)$. При этом следует иметь в виду, что для каждой вершины i' – спусковой функции соответствующие переменные $x_{i'}$ будут определены на доменах $D_i = \{m_0\}$, содержащих один нулевой ресурс. Данный факт существенно сокращает пространство поиска.

3) $R \subseteq (D_1 \times D_2 \times \ldots \times D_{n'})$.

4) C – множество ограничений, которое делится на несколько групп.

Множество ограничений представляет собой в основном множество рекурсивных функций производных от рекурсивных функций операций. Модификации вызваны переходом к развернутому графу и связанной с этим заменой переменных. Данные функции порождают ограничения, обусловленные допустимыми расписаниями для каждой операции конвейера при некотором заданном распределении ресурсов для пар (операция, заказ). Последнее условие проверяет допустимость расписания с точки зрения того, чтобы ресурс в некоторый момент времени не использовался двумя операциями.

1-я группа — это множество унарных ограничений, определенных для начальных операций ($1 \leqslant i \leqslant n_0$) и построенных на основе функции (2).

$$bop(i,k) = \begin{cases} p_{ij}, \\ \qquad \textbf{если } (x_{i'} = m_j)\&(k = 0); \\ bop(i,k-1) + p_{ij}, \\ \qquad \textbf{если } (x_{i'} = x_{pk(i')})\&(x_{i'} = m_j)\&(0 < k \leqslant \hat{k}); \\ bop(i,k-1) - p_{ij} + p_{il}, \\ \qquad \textbf{если } (x_{i'} \neq x_{pk(i')})\&(x_{i'} = m_l)\&(x_{pk(i')} = m_j)\&(0 < k \leqslant \hat{k}). \end{cases}$$

2-я группа — это множество бинарных ограничений, определенных для пары вершин графа, соединенных дугой и построенных на основе функций (3), (5)–(8).

$$op(i,k) = \begin{cases} t(pred(i),k) + p_{ij}, \\ \qquad \textbf{если } (type(i) = \textbf{op})\& \\ \qquad\qquad \&(x_{i'} = m_j)\&(k = 0); \\ t(pred(i),k) + p_{ij}, \\ \qquad \textbf{если } (type(i) = \textbf{op})\&(t(pred(i),k) \geqslant t(i,k-1)\& \\ \qquad\qquad \&(x_{i'} = m_j)\&(0 < k \leqslant \hat{k}); \\ t(pred(i),k) + p_{ij}, \\ \qquad \textbf{если } (type(i) = \textbf{op})\&(t(pred(i),k) < t(i,k-1)\& \\ \qquad\qquad \&(x_{i'} \neq x_{pk(i')})\&(x_{i'} = m_j)\&(0 < k \leqslant \hat{k}); \\ t(i,k-1) + p_{ij}, \\ \qquad \textbf{если } (type(i) = \textbf{op})\&(t(pred(i),k) < t(i,k-1)\& \\ \qquad\qquad \&(x_{i'} = x_{pk(i')})\&(x_{i'} = m_j)\&(0 < k \leqslant \hat{k}). \end{cases}$$

$$mul(i,k) = t(pred(i), \lfloor k/q_i \rfloor), \ \textbf{если } (type(i) = \textbf{mul})\&(0 \leqslant k \leqslant \hat{k});$$

$$red(i,k) = t(pred(i), (k+1)q_i - 1), \ \textbf{если } (type(i) = \textbf{red})\&(0 \leqslant k \leqslant \hat{k});$$

$$get1(i,k) = t(pred(i), 2k), \\ \textbf{если } (type(i) = \textbf{get1})\&(k \bmod 2 = 0)\&(0 \leqslant k \leqslant \hat{k});$$

$$get2(i,k) = t(pred(i), 2k+1), \\ \textbf{если } (type(i) = \textbf{get2})\&(k \bmod 2 = 1)\&(0 \leqslant k \leqslant \hat{k}).$$

3-я группа — это множество ограничений, определенных для тройки вершин графа и построенных в соответствии с функциями (4) и (9).

$$and(i,k) = \max(t(pred1(i),k), t(pred2(i),k)), \quad \textbf{если } (type(i) = \textbf{and}).$$

$$put(i,k) = \begin{cases} t(pred1(i),k), \\ \qquad \textbf{если } (type(i) = \textbf{put})\&(k=0); \\ \max(put(i,k-1), t(pred2(i),k/2)), \\ \qquad \textbf{если } (type(i) = \textbf{put})\& \\ \qquad\quad \&(k \bmod 2 = 0)\&(0 < k \leqslant \hat{k}); \\ \max(put(i,k-1), t(pred1(i),(k-1)/2)), \\ \qquad \textbf{если } (type(i) = \textbf{put}) \\ \qquad\quad \&(k \bmod 2 = 1)\&(0 < k \leqslant \hat{k}). \end{cases}$$

4-я группа ограничений относится к ограничениям типа **all-different**. Она отображает тот факт, что использование ресурса в каждый момент времени возможно не более чем одной операцией. Введем следующие обозначения:
$\tau_{i'} = (t^b_{i'}, t^e_{i'})$ обозначает интервал времени от $t^b_{i'}$ до $t^e_{i'}$ $(t^b_{i'} \leqslant t^e_{i'})$, в течение которого выполняется i-й операцией k-й заказ $(i = \psi(i'),\ k = \varphi(i'))$ и используется некоторый ресурс m_j. Операция $\cap$ принимает значение истины, если временные отрезки пересекаются.

$$\tau_{i'} \cap \tau_{i''} = \begin{cases} true, \text{ если } (t^b_{i'} < t^e_{i''} \leqslant t^e_{i'}) \ or \ (t^b_{i'} \leqslant t^b_{i''} < t^e_{i'}); \\ false, \ otherwise. \end{cases}$$

В этих определениях глобальное ограничение будет выглядеть следующим образом:

$$\begin{aligned} \forall i' \forall i'' \ (1 \leqslant i', i'' \leqslant n')\&(i' \neq i'')\&(\tau_{i'} \cap \tau_{i''}) \vdash \\ \vdash ((type(i') \neq \textbf{bop})\&(type(i') \neq \textbf{op})) \vee \\ \vee (type(i'') \neq \textbf{bop})\&(type(i'') \neq \textbf{op})) \vee \\ \vee (x_{i'} \neq x_{i''}). \end{aligned}$$

Смысл условия в том, что если интервалы двух разных вершин i' и i'' пересекаются, то либо хотя бы одна из них соответствует спусковой функции, либо они используют разные ресурсы.

Решением ЗУО будет нахождение таких значений переменных $x_{i'}$, при которых все перечисленные условия будут удовлетворены, и очевидно, что в данной постановке ЗУО всегда будет иметь некоторое решение. Для любого допустимого распределения ресурсов можно построить расписание. В статье требуется оптимальное расписание. Пусть T_{opt} – время выполнения оптимального расписания. Будем полагать, что решение квазиоптимальное с точностью до некоторой, наперед заданной величины Δ, если найденное решение T такое, что $T - T_{opt} \leqslant \Delta$. Вычисление квазиоптимального значения времени выполнения расписания можно осуществить с помощью следующего известного алгоритма:

1) Выбрать некоторое множество значений V, равное V_0.
Вычислить значение $T_{\max} = t(n', \hat{k})$ для V_0.
Положить $T_{\min} = 0$.

2) Если $T_{\max} - T_{\min} \leqslant \Delta$, то задача решена и $T_{\max}$ является решением, а соответствующее данному решению множество V является искомым распределением ресурсов.

3) Вычислить значение $T = T_{\min} + (T_{\max} - T_{\min})/2$.

4) Ввести в ЗУО дополнительное ограничение: $t(n', \hat{k}) \leqslant T$.

5) Если ЗУО с дополнительным условием не будет иметь решение, то положить $T_{\min} = T$ и перейти к п. 3.

6) Если ЗУО с дополнительным условием будет иметь решение, то положить $T_{\max} = T$ и перейти к п. 2.

Конец алгоритма.

Найденное значение $T_{\max}$ является квазиминимальным, но квазиоптимальность этого значения следует доказать. Суть проблемы состоит в том, что рекурсивные функции на локальном уровне строят оптимальное расписание (реализуют жадный алгоритм), и следует доказать, что при фиксированном распределении ресурсов локальная оптимизация является и глобальной.

Утверждение 1. Для любого допустимого значения $V = \{x_1, \ldots, x_{n'}\}$ *рекурсивная функция* $t(i, k)$ *вычисляет одно из оптимальных значений для* $1 \leqslant i \leqslant n$ *и* $0 \leqslant k \leqslant \hat{k}$.

Доказательство осуществим методом математической индукции.

Для системности изложения напомним формулировку принципа математической индукции. Некоторое утверждение $P(i)$, зависящее от натурального параметра i, считается доказанным, если:

1) установлено, что $P(1)$ верно;

2) для любого $1 \leqslant i < n'$ из предположения, что $P(i)$ верно, следует, что $P(i+1)$ верно.

Применим данный метод к развернутому графу. Из теории графов известно [17], что ациклический граф является строго частично упорядоченным. Существуют алгоритмы построения для строго частично упорядоченного графа некоторого линейного порядка. Неформально линейный порядок — это когда все вершины графа расположены в строку и все дуги направлены слева направо. Пример построения линейного порядка из строго частичного показан на рис. 9, где *а*) — это исходный ациклический граф, а *б*) — один из возможных линейных порядков. В данной статье устроит любой такой алгоритм с условием, что первые n_0 вершин исходного графа так и останутся первыми, хотя можно и в другом порядке, а вершина n останется последней. При вычислении рекурсивной функции в соответствии с линейным порядком ее аргументы уже будут вычислены.

Метод математической индукции применим следующим образом.

Шаг 1. Исходя из определения рекурсивного конвейера и способа упорядочивания вершин исходного графа очевидно, что все начальные вершины

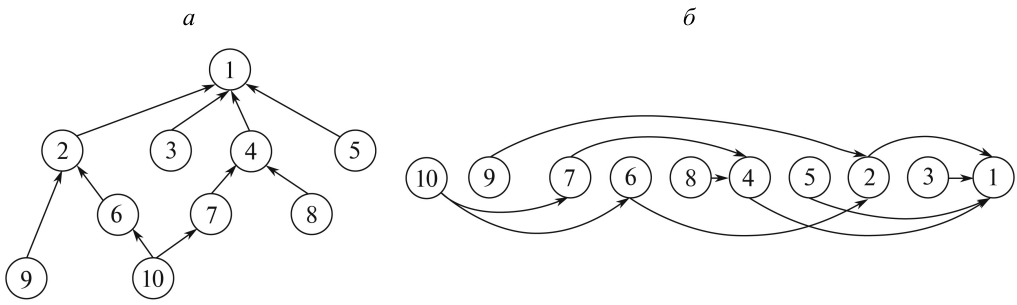

Рис. 9. Пример линейного порядка графа.

исходного графа являются операциями (не функциями) и являются производными для начальных вершин развернутого графа. Если в исходном графе n_0 начальных вершин, то и в развернутом графе тоже n_0 начальных вершин и для каждой начальной вершины i в исходном графе существует вершина $(0, i)$ в развернутом графе, которой не предшествует никакая другая вершина. Если $1 \leqslant i \leqslant n_0$ – некоторая начальная вершина исходного графа, то i' такая, что $\psi(i') = i$ и $\varphi(i') = 0$ является начальной вершиной развернутого графа.

Таким образом, для каждой начальной вершины $i' \to (i, k)$ развернутого графа справедливы следующие утверждения:

$$type(i) = \mathbf{bop},$$
$$1 \leqslant \psi(i') \leqslant n_0 \quad \text{и} \quad \varphi(i') = 0, \quad \text{т.е.} \quad (1 \leqslant i \leqslant n_0)\&(k = 0).$$

В этом случае в соответствии с функцией (1)

$$t(i, 0) = bop(i, 0) = p_{ij}, \quad \text{если} \quad x_{i'} = m_j.$$

Таким образом, для конкретного значения $x_{i'}$ значение $t(i, 0)$ единственно и, следовательно, оно оптимально. Утверждение верно. Поскольку данное утверждение верно для любой из n_0 начальных операций, далее эти варианты рассматривать не будем.

Шаг 2. Допустим, что для вершины любого номера $n_0 < i' < n'$ утверждение верно. Докажем, что в этом случае оно верно и для вершины с номером $i'' = i' + 1$. В соответствии с (1) возможны 8 вариантов.

Рассмотрим варианты 1 и 2, когда

$$type(i'') = \mathbf{bop} \quad \text{и}$$
$$type(i'') = \mathbf{op}.$$

Вариант 1. Выполняется начальная операция над ненулевым заказом, т.е. $i'' \to (i, k)$ и $(1 \leqslant i \leqslant n_0)\&(k > 0)$. Если $x_{i''} = x_{pk(i'')}$, то операция i при выполнении k-го и $(k-1)$-го заказов использует один ресурс.

В этом случае время завершения вычисляется по формуле (2,а) (диаграмма на рис. 10,*а*). Если же используются разные ресурсы, то выполнение операции i можно осуществить одновременно над заказами $(k-1)$ и k. Завершение выполнения вычисляется по формуле (2,б) (диаграмма на рис. 10,*б*). Длительности операций могут не совпадать. Учитывая, что время выполне-

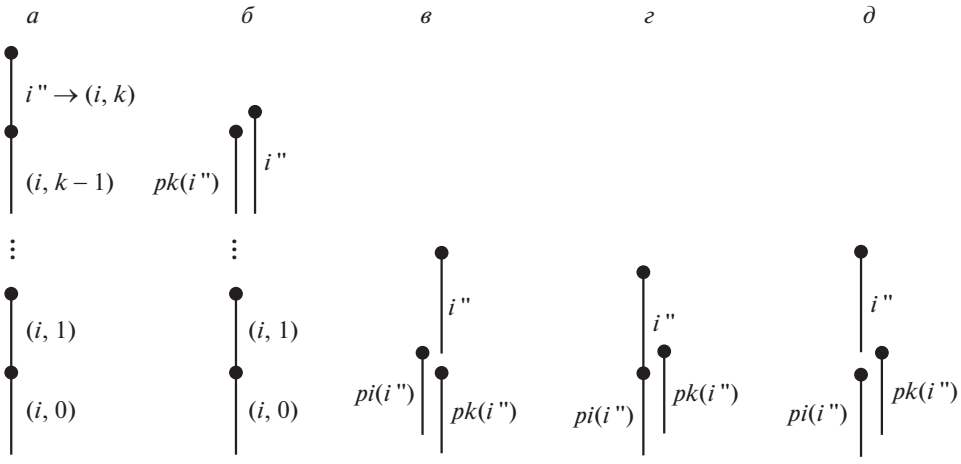

Рис. 10. Варианты фрагментов временных диаграмм.

ния операцией i $(k-1)$-го заказа оптимально, время выполнения i k-го заказа также оптимально. Вариант рассмотрен.

Вариант 2. В этом случае выполняется неначальная операция. Если $k = 0$, то время завершения вычисляется по формуле (3,а). Если $k > 0$, то время завершения вычисляется по вариантам (3,б)–(3,г). Соответствующие примеры временных диаграмм приведены на рис. 10,*c–e*. Во всех вариантах новое расписание с добавлением вершины i'' будет оптимальным, если предшествующее расписание было оптимальным.

Варианты 3–8. Во всех данных вариантах вершины соответствуют спусковым функциям. В этом случае не производится распределение ресурсов и итоговое расписание строится единственным возможным способом исходя из отношения временного предшествования, определяемого соответствующей рекурсивной функцией.

Утверждение доказано.

Временная сложность алгоритма равна

$$O\left(\log_2\left(\frac{T_0}{\Delta}\right)\prod_{i=1}^{nk} \mid D_i \mid\right),$$

где T_0 – значение времени выполнения расписания для некоторого произвольного множества значений V_0, Δ – некоторая наперед заданная константа точности вычисления результата, а $\mid D_i \mid = 1$ для спусковых функций.

5. Заключение

В данной статье осуществлена теоретическая постановка задачи. Впервые для исследования конвейерной модели используется такой инструмент, как Constraint Programming. Применение данного метода на практике является дальнейшим направлением работы авторов статьи.

В настоящее время существует большое количество коммерческих и свободного использования систем *программирования в ограничениях*. Обзор таких систем можно посмотреть в [2].

Изучение вычислительной сложности решения ЗУО представляется фундаментальной проблемой. В [18] было показано, что класс всех ЗУО является NP-трудным, так что вряд ли существуют эффективные (полиномиальные) алгоритмы общего назначения для решения всех видов ЗУО. Классы легко разрешимых ЗУО, которые могут быть решены за полиномиальное время, обычно описываются или деревьями и древовидными графовыми структурами [19, 20], или определенной комбинацией алгебраических операторов [21]. Рассматриваемая в статье задача базируется на дереве ограничений (исходный ациклический граф и расширенный граф), что внушает оптимизм по поводу существования эффективного алгоритма решения поставленной задачи. Построение расписания для конвейера вычислением суперпозиции рекурсивных функций предполагает для решения ЗУО вариант *поиска с возвратами* (backtracking) в различных модификациях [2]. Важно отметить, что введение в модель конвейера спусковых функций нисколько не увеличивает алгоритмическую сложность решения задачи.

Использование данного метода дает два важных преимущества.

Первое. Метод оптимизации рекурсивного конвейера, рассмотренный в статье [22], сводится к задаче целочисленного линейного программирования и рассматривает строго определенный набор рекурсивных функций. Введение новых функций требует дополнительного рассмотрения корректности применения метода. Подход, используемый в данной статье, хотя и рассматривает конкретный набор функций, однако никаких ограничений, кроме вычислимости, к ним не предъявляет. Это позволяет вводить в рассмотрение новые рекурсивные функции без обсуждения вопроса корректности применения метода.

Второе. Предложенный метод позволяет распределять ресурсы без их закрепления за отдельными операциями. Данный подход увеличивает размерность задачи, но открывает возможности для построения более эффективного расписания.

СПИСОК ЛИТЕРАТУРЫ

1. *Лазарев А.А., Гафаров Е.Р.* Теория расписаний. Задачи и алгоритмы. М.: Изд-во МГУ, 2011.
2. *Rekiek B., Dolgui A., Delchambre A., Bratcu A.* State of art of optimization methods for assembly line design // Annual Reviews in Control. 2002. V. 26. P. 163–174.
3. *Ghosh S., Gagnon R.J.* A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems // Int. J. Product. Res. 1989. V. 26. No. 4. P. 637-0670.
4. *Helgeson W.B., Salveson M.E., Smith W.W.* How to balance an assembly line / Technical Report, Carr Press. 1954. New Caraan, Conn.
5. *Bowman E.H.* Assembly line balancing by linear programming // Oper. Res. 1960. No. 8(3). P. 3850–389.

6. *Salveson M.E.* The assembly line balancing problem // J. Indust. Engineer. 1955. No. 6. P. 18–25.
7. *Neumann K., Schwindt C., Zimmermann J.* Recent results on resource-constrained project scheduling with time windows: Models, solution methods, and applications // Central Eur. J. Oper. Res. 2002. V. 10. 2. P. 113–148.
8. *Drexl A., Kimms A.* Optimization guided lower and upper bounds for the resource investment problem // J. Oper. Res. Society. 2001. V. 52. No. 3. P. 340–351.
9. *Ranjbar M., Kianfar F., Shadrokh S.* Solving the resource availability cost problem in project scheduling by path relinking and genetic algorithm // Applied Mathematics and Computation. 2008. V. 196. No. 2. P. 879–888.
10. *Yamashita D.S., Armentano V.A., Laguna M.* Robust optimization models for project scheduling with resource availability cost // Journal of Scheduling. 2007. V. 10. No. 1. P. 67–76.
11. *Щербина О.А.* Удовлетворение ограничений и программирование в ограничениях// Интеллектуальные системы. 2011. Т. 15. № 1–4. С. 53–170.
12. *Apt K.R.* Principles of Constraint Programming. New York: Cambridge University Press, 2003.
13. *Нариньяни А.С., Седреева Г.О., Седреев С.В., Фролов С.А.* TimeEX/Windows – новое поколение недоопределенной технологии календарного планирования / Проблемы представления и обработки не полностью определенных знаний. Под ред. И.Е. Швецова. Москва; Новосибирск, 1996. С. 101–116.
14. *Куприянов Б.В.* Рекурсивные конвейерные процессы – основные свойства и характеристики // Экономика, статистика и информатика. Вестн. УМО. 2015. № 1. С. 163–170.
15. *Куприянов Б.В.* Применение модели конвейерных процессов рекурсивного типа для решения прикладных задач // Экономика, статистика и информатика. Вестник УМО. 2014. № 5. С. 170–179.
16. *Куприянов Б.В.* Метод эффективного анализа модели рекурсивного конвейерного процесса // АиТ. 2017. № 3. С. 63–79.
    Kupriyanov B.V. Method of Efficient Analysis of the Recursive Conveyor Process Models // Autom. Remote Control. 2017. V. 78. No. 3. P. 435–449.
17. *Оре О.* Теория графов. М.: Наука, 1980.
18. *Mackworth A.K.* Consistency in networks of relations // Artificial Intelligence. 1977. V. 8. No. 1. P. 99–118.
19. *Dechter R., Pearl J.* Tree clustering for constraint networks // Artificial Intelligence. 1989. V. 38. No. 3. P. 353–366.
20. *Freuder E.C.* Backtrack-free and backtrack-bounded search / Search in Artificial Intelligence / L. Kanal, V. Kumar (eds.). Springer-Verlag, 1988. P. 343–369.
21. *Jeavons P.G., Cohen D.A., Gyssens M.* Closure properties of constraints // Journal of ACM. 1997. V. 44. P. 527–548.
22. *Куприянов Б.В.* Оценка и оптимизация производительности рекурсивного конвейера // АиТ. 2020. № 5. С. 6–25.
    Kupriyanov B.V. Estimation and Optimization of the Efficiency of a Recursive Conveyor // Autom. Remote Control. 2020. V. 81. P. 775–790.

Статья представлена к публикации членом редколлегии А.А. Галяевым.

Поступила в редакцию 25.01.2021
После доработки 21.06.2021

Принята к публикации 30.06.2021