

Optimization of a Recursive Conveyor by Reduction to a Constraint Satisfaction Problem

B. V. Kupriyanov^{1*} and A. A. Lazarev^{1**}

¹*Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, Moscow, 117997 Russia*
*e-mail: *kupriyanovb@mail.ru, **jobmath@mail.ru*

Received January 25, 2021; revised June 21, 2021; accepted June 30, 2021

Abstract—We consider the recursive conveyor schedule optimization problem. To this end, we introduce the definition of a conveyor described by a connected acyclic graph, in which each vertex is an operation or a control function associated with the corresponding recursive function from a certain finite set. Each recursive function defines a precedence relation for a conveyor operation. The solution of the problem of minimizing the time of order fulfillment by a conveyor on a finite set of renewable resources is considered. The solution is carried out by reduction to the constraint satisfaction problem.

Keywords: scheduling theory, conveyor balancing, flow-shop problem, constraint satisfaction problem

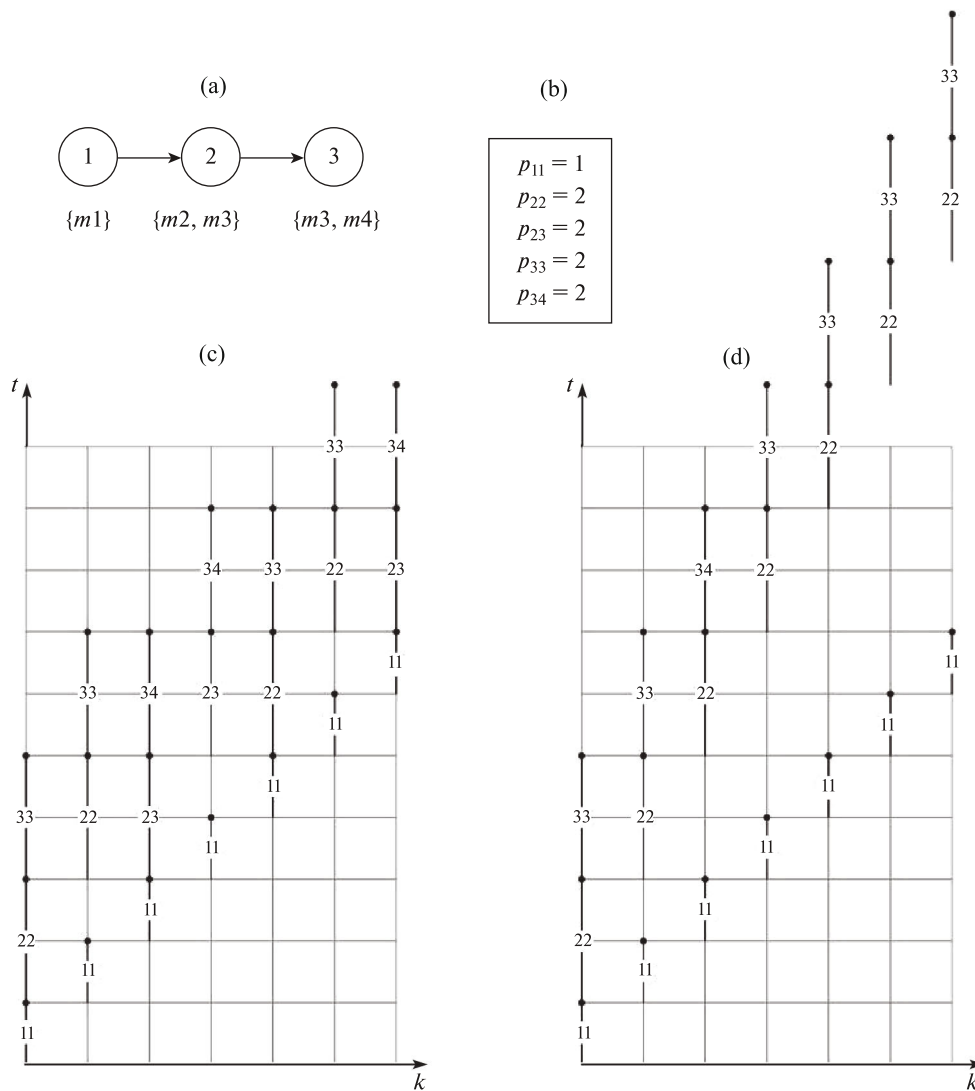
DOI: 10.1134/S0005117921110059

1. INTRODUCTION

Resource-Constrained Project Scheduling Problem (RCPSp) is one of the main problems in scheduling theory. In RCPSp, a set of orders is specified that must be served on a finite set of machines (hereinafter referred to as resources). An order is identical to a set of ordered operations and is characterized by the execution time. It is necessary to minimize the lead time for the set of resource allocations across operations. This type of problems is also relevant for conveyor systems [1]; see [2] for a list of publications that addresses assembly line balancing and resource scheduling techniques during the conveyor design phase. The paper [3] provides a comprehensive overview and analysis of various methods for designing and planning conveyor systems. Most papers deal with assembly line balancing (ALB). Methods for finding optimal solutions using linear [4, 5] and integer [6] programming are proposed for the ALB model. The papers [7–10] are of interest among contemporary papers on resource scheduling optimization for typical problems of scheduling theory.

It is important to note two features in the statement of such problems.

The first is that resources are allocated prior to the order fulfillment process and to the entire fulfillment process. This results in each resource being assigned to a single operation, even if it could potentially be used by multiple operations. However, it can be shown by example that transferring a resource from one operation to another can lead to a more efficient schedule. Figure 1a shows a model of a conveyor of three sequential operations. Potentially used resources from the set $\{m_1, m_2, m_3, m_4\}$ are indicated in curly braces under the operations. Figure 1b shows the duration of operations when using different resources. The first index is the operation number, and the second is the resource number. Figures 1c and 1d show the timing diagrams of operations for seven orders. The abscissa is the order number, and the ordinate is time. The segment with index i, j defines the time interval for the i th operation using the j th resource for the k th order. Figure 1d shows an optimal diagram with resource assignment to an operation for the entire duration



of orders, and Fig. 1c shows a diagram with resource allocation without assignment to operations. Comparison of the diagrams shows the time advantage of the second method.

The second feature is the use of a limited set of precedence relationships. This is a relation of precedence between operations and it is defined using the “and” predicate.

The present paper proposes a solution to overcome both of these constraints. The resource distribution with the possibility of using one resource by several operations and with the expansion of precedence relations using recursive functions is considered. Constraint Satisfaction Problems (CSPs) and Constraint Programming methods [11–13] are used, among other things, to solve scheduling problems. In this paper, we consider the solution of the RCPSP by reducing it to a CSP for conveyors described by recursive functions [14]. The execution time of operation i depends on the used resource j , i.e., is equal to p_{ij} ; see [15] for examples of capabilities and use of such conveyors.

The paper is organized as follows. Section 2 defines a recursive conveyor and describes recursive functions in detail. Section 3 provides examples of recursive conveyors. Section 4 describes the definition of CSP and states the problem of allocating conveyor resources as a CSP. Section 5

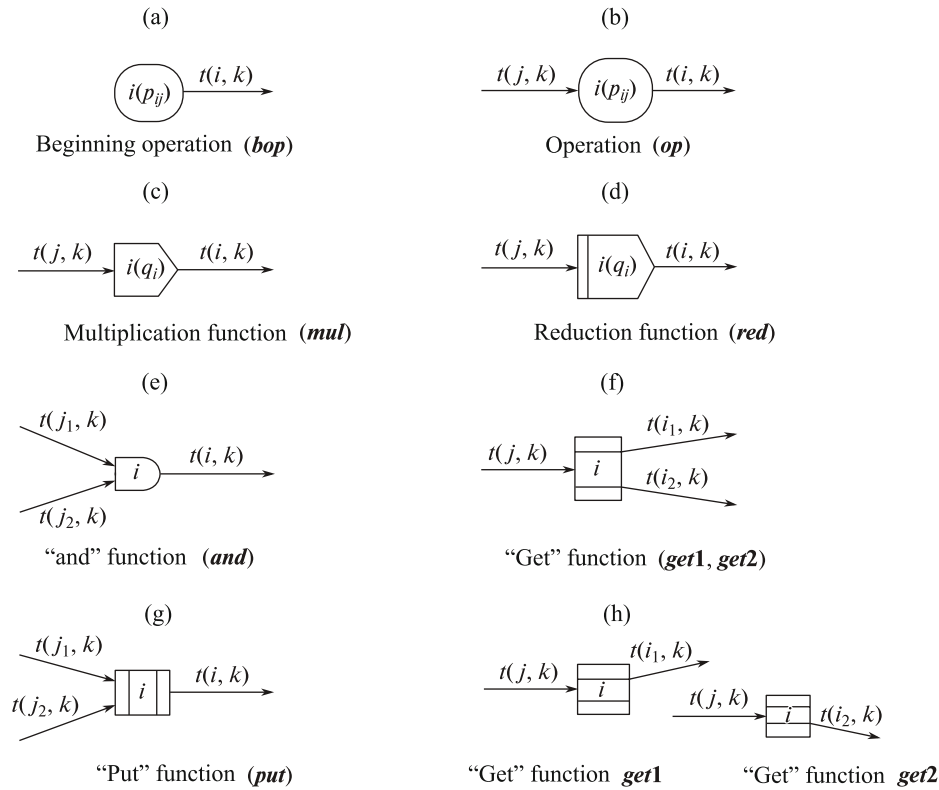


Fig. 2. Graphic notation of operations and trigger functions of a conveyor.

contains conclusions and examines the problem on the complexity of the algorithm for solving the CSP stated in the paper.

2. DEFINITION OF CONVEYOR

The model and properties of a recursive conveyor are discussed in detail in [14, 16]. However, they are described under the assumption that each operation has its own resource assigned to it for the duration of execution of all orders. In this section, the definition of recursive conveyor functions is given taking into account the resource allocation for each (operation, order) pair.

The conveyor model is a connected acyclic directed graph $G = (V, A)$ with a single terminal vertex; V is the set of graph vertices and n is the number of vertices, and A is the set of graph edges—ordered pairs of the form (v, w) , where $v, w \in V$. The vertices of the graph are labeled by numbers from the set $I = \{1, \dots, n\}$ so that the first n_0 ($1 \leq n_0 < n$) vertices be initial and the vertex n be terminal. The visual image of the vertices and edges of the graph is presented in Fig. 2. Here i, j, j_1, j_2 are the numbers of operations (vertices), k is the order number, $p_{i,j}$ is the time of execution of the i th operation when using the j th resource, and q_i is the multiplication or reduction coefficient. Each vertex of the graph may have one input edge (see Figs. 2b, 2c, 2d, 2f) or two edges (see Figs. 2e, 2g) depending on the vertex type. The predecessor functions $pred, pred1, pred2 : I \rightarrow I$ are defined on the set of numbers of graph vertices.

The function $pred(i)$ is defined for a vertex that has one predecessor vertex and calculates the number j of the graph vertex such that $(j, i) \in A$.

The following two functions are defined for vertices that have two predecessor vertices, called *vertex 1* and *vertex 2*:

$pred1(i)$ calculates the number j_1 of the graph *vertex 1* such that there exists an edge $(j_1, i) \in A$.

$pred2(i)$ calculates the number j_2 of the graph *vertex 2* such that there exists an edge $(j_2, i) \in A$.

Each graph vertex may correspond to an operation or a trigger function. The operations are associated with production operations. The trigger functions define the relations of time precedence of operations. The graph vertices are labeled using the mapping

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

to the set of eight elements. Each constant of the set E denotes the type of a graph vertex, and the vertex has the corresponding graphic notation (Fig. 2). The conveyor uses the set of resources $M = \{m_1, \dots, m_r\}$. Each operation i may potentially use resources from some set $D_i \subseteq M$. For each $1 \leq i \leq n$, we define a variable x_{ik} on the set D_i . If $x_{ik} = m_j$, then the operation i uses resource $m_j \in D_i$ when executing the k th order. The execution time of the operation is p_{ij} and is a constant. At each time, each resource can be used by only one operation. Interruptions of operations are forbidden.

Trigger functions do not consume resources, and their execution time is 0.

The operation execution schedule is constructed using recursive functions $R : I \times K \times M \rightarrow T$, where K is a finite set of order numbers and T is time.

For example, the recursive function

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

or

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

where $j = \text{pred}(i)$, $j_1 = \text{pred1}(i)$, $j_2 = \text{pred2}(i)$ calculates the completion time of processing the k th order by the i th operation for $k \in K$ based on the time when it completes the $(k-1)$ st order and on the completion times of the k th order by the preceding operations. The recursive functions have two arguments: i is the operation number (the number of a graph vertex), and k is the order number. There is a specific recursive function associated with each type of a vertex, i.e., with each element of the set E .

The recursive function $t(i, k)$ calculated using formula (1) implements the superposition of the recursive functions corresponding to graph vertices and calls one of them in accordance with the vertex type,

$$t(i, k) = \begin{cases} bop(i, k) & \text{if } type(i) = \mathbf{bop} \\ op(i, k) & \text{if } type(i) = \mathbf{op} \\ and(i, k) & \text{if } type(i) = \mathbf{and} \\ mul(i, k) & \text{if } type(i) = \mathbf{mul} \\ red(i, k) & \text{if } type(i) = \mathbf{red} \\ get1(i, k) & \text{if } type(i) = \mathbf{get1} \\ get2(i, k) & \text{if } type(i) = \mathbf{get2} \\ put(i, k) & \text{if } type(i) = \mathbf{put}. \end{cases} \quad (1)$$

The calculation of the completion time of the k th order by the conveyor is performed by calling the recursive function $t(n, k)$.

The following describes all types of nodes, including their purpose, type, and the corresponding recursive function.

1. The initial production operation (see *bop* in Fig. 2a) with number $1 \leq i \leq n_0$ is characterized by the completion time p_{ij} of the k th order if the operation uses the resource m_j ($x_{ik} = m_j$). The completion time of this operation for the k th order is determined as the completion time of the $(k-1)$ st order plus the execution time of the operation p_{ij} . If an operation involved in the execution of the k th and $(k-1)$ st order uses different resources, then their completion times are matched. The execution time of the zero order is equal to p_{ij} ,

$$\begin{aligned}
 & bop(i, k) : bop \\
 & = \begin{cases} p_{ij} & \text{if } (x_{ik} = m_j) \& (k=0) \\ bop(i, k-1) + p_{ij} & \text{if } (x_{ik} = x_{i,k-1}) \& (x_{ik} = m_j) \& (k > 0) \\ bop(i, k-1) + p_{ij} - p_{il} & \text{if } (x_{ik} \neq x_{i,k-1}) \& (x_{i,k-1} = m_l) \& (x_{ik} = m_j) \& (k > 0). \end{cases} \quad (2)
 \end{aligned}$$

2. A noninitial production operation (see *op* in Fig. 2b) with number $n_0 < i$ is characterized by the execution time of p_{ij} if resource m_j is used. The completion time of the k th order by this operation is determined as the maximum of the completion times of the $(k-1)$ st order by this operation and of the k th order by the preceding operation plus the execution time of the operation p_{ij} . In this case, the beginnings of execution of the k th and $(k-1)$ st orders by the i th operation are matched if different resources are used for its execution. The execution time of the zero order is equal to the execution time of the zero order by the preceding operation plus p_{ij} ,

$$\begin{aligned}
 & op(i, k) : op = \begin{cases} t(pred(i), k) + p_{ij} & \text{if } (x_{ik} = m_j) \& (k=0) \\ t(pred(i), k) + p_{ij} & \text{if } (t(pred(i), k) \geq t(i, k-1)) \& (x_{ik} = m_j) \& (k > 0) \\ t(pred(i), k) + p_{ij} & \text{if } (t(pred(i), k) < t(i, k-1)) \\ & \& (x_{ik} \neq x_{i,k-1}) \& (x_{ik} = m_j) \& (k > 0) \\ t(i, k-1) + p_{ij} & \text{if } (t(pred(i), k) < t(i, k-1)) \\ & \& (x_{ik} = x_{i,k-1}) \& (x_{ik} = m_j) \& (k > 0). \end{cases} \quad (3)
 \end{aligned}$$

3. The trigger function *and* (see *and* in Fig. 2e) calculates the completion time of the k th order for two preceding operations,

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

4. The trigger multiplication function (see *mul* in Fig. 2c) for each completion time by the preceding operation calculates q_i completion times of operation i . This means that for one execution of the operation preceding to i there are q_i ($q_i \geq 1$) executions of operation i ,

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

where $\lfloor x \rfloor$ denotes the integer part of x .

5. The trigger reduction function (see *red* Fig. 2d) is the inverse function of the multiplication function and calculates the completion time of the next batch of q_i ($q_i \geq 1$) orders,

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

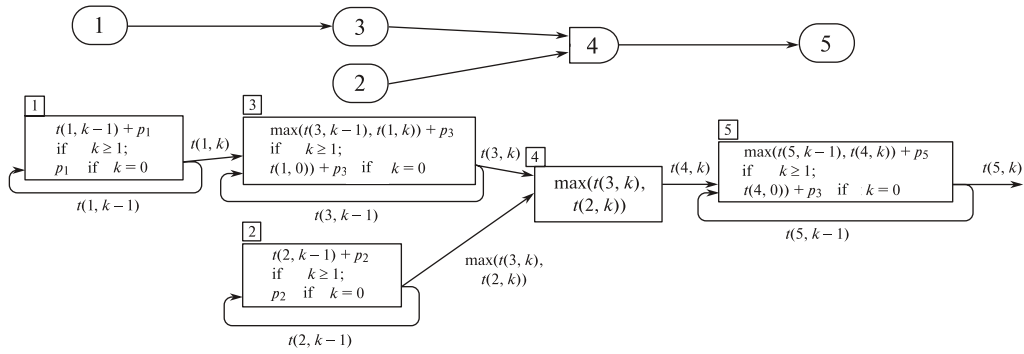


Fig. 3. Example of conveyor graph (top) and the corresponding graph of superposition of recursive functions (bottom).

6. The trigger get function (see *get* in Fig. 2f) simulates splitting conveyor operations into two threads. For a user it serves as one function, and when realized, it splits into two (conditionally, upper and lower), see Fig. 2h. For the upper variant, the trigger function *get1* calculates the completion times of even orders; i.e., $t(i, 0) = t(p, 0)$, $t(i, 1) = t(p, 2)$, $t(i, 2) = t(p, 4), \dots$,

$$get1(i, k) : get1 = t(pred(i), 2k), \quad k \geq 0. \quad (7)$$

For the lower one, it calculates the completion times of odd orders; i.e., $t(j, 0) = t(p, 1)$, $t(j, 1) = t(p, 3)$, $t(j, 2) = t(p, 5), \dots$,

$$get2(i, k) : get2 = t(pred(i), 2k + 1), \quad k \geq 0. \quad (8)$$

7. The trigger put function (see *put* in Fig. 2g) is the inverse of the get function and simulates merging two conveyors into one. The easiest way to explain this is by the sequence of values $t(i, k)$: $t(i, 0) = t(p, 0)$, $t(i, 1) = t(q, 0)$, $t(i, 2) = t(p, 1)$, $t(i, 3) = t(q, 1)$, $t(i, 4) = t(p, 2)$, $t(i, 5) = t(q, 2), \dots$,

$$put(i, k) : put = \begin{cases} t(pred1(i), k) & \text{if } k = 0 \\ \max(put(i, k-1), t(pred2(i), k/2)) & \text{if } (k \bmod 2 = 0) \& (k > 0) \\ \max(put(i, k-1), t(pred1(i), (k-1)/2)) & \text{if } k \bmod 2 = 1. \end{cases} \quad (9)$$

Here $x \bmod y$ is the remainder of the division of x by y .

Figure 3 shows an example of a conveyor graph and the corresponding superposition of recursive functions for the case where each operation has one resource allocated for the entire process duration. In reality, such a graph is not constructed—it results from the process of calculation of the recursive function $t(i, k)$.

The start time of the conveyor operation is set equal to 0.

In the proposed model, the interpretation of the operations and the trigger function *and* does not differ from those in scheduling theory. The remaining set of trigger functions has been developed in practice based on model construction and is represented by two pairs of direct and inverse functions (*mul* – *red* and *get* – *put*). It is possible to construct new functions extending the precedence relation.

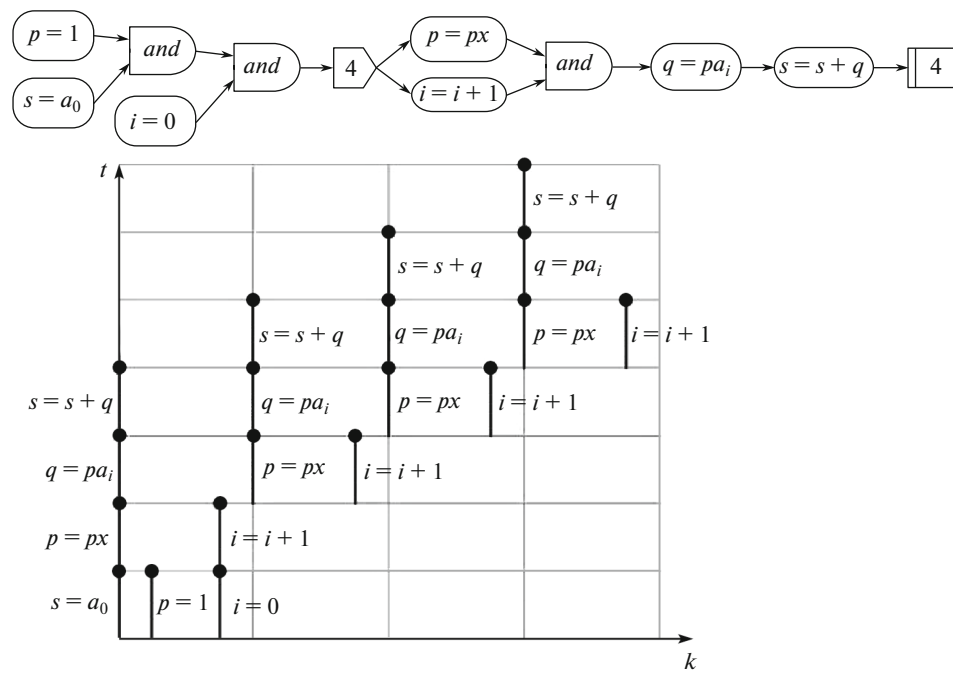


Fig. 4. Example of a computational conveyor model and a timing diagram.

3. EXAMPLES OF MODELS

Examples of some conveyor models for various applications are given in [15]. The present paper provides additional examples of recursive conveyors. Before giving the examples, we note that the time diagram should be drawn up for each operation separately, but this is unclear and cumbersome, and so the diagrams will be combined in one. The abscissa will show the order numbers, and the ordinate will show the time. With such a combination, some time intervals will merge in one diagram. To prevent this from happening, the segments with the value k on the abscissa axis will be placed in the interval between divisions k and $(k + 1)$.

Example 1. Figure 4 presents the model of a conveyor that calculates the polynomial

$$p = a_0 + a_1x^1 + a_2x^2 + a_3x^3 + a_4x^4$$

and the corresponding timing diagram under the assumption that each operation is performed by its own processor. The execution times are equal to 1 for all operations.

Example 2. Before considering the second example, let us look at an auxiliary one. Figure 5 provides an example of a conveyor and its timing diagram. The conveyor consists of three operations, a reduction function with factor 2 and a multiplication function with factor 3. The diagram shows that all operations are performed with different multiplicities, but the process is a conveyor belt. To understand that such a conveyor can take place in practice, consider an example of a conveyor assembly of some product that has the structure shown in Fig. 6. The specific feature of the assembly is that unit 2 and unit 3 are assembled in a place remote from the main unit assembly. Therefore, a batch of q_2 units is assembled on site and then transported to the warehouse of the main production. Similarly, unit 3 is assembled at another remote location and transported in batches of q_3 units. It is required to draw up a schedule for production of units, their storage in a warehouse, and transportation and assembly of the main product as a single conveyor process. An example of a model of such a process is shown in Fig. 7. Its essence is that unit 2 accumulates

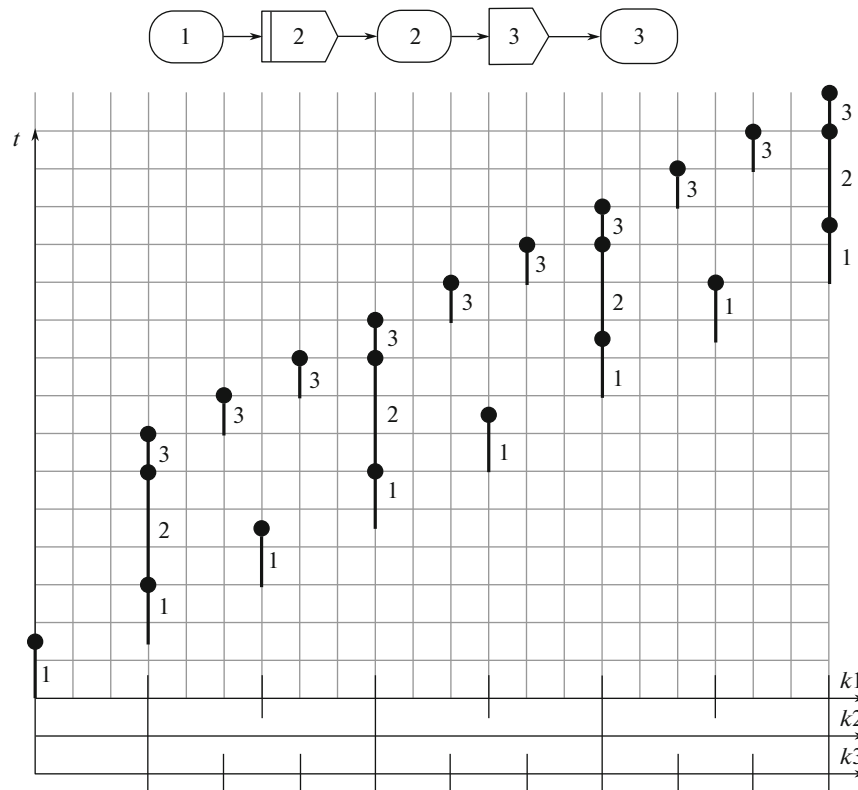


Fig. 5. Example of a simple conveyor with multiplication and reduction functions.

in the warehouse in the amount of q_2 pieces as the assembly line progresses, and then this batch is delivered to the main assembly site. The pipelining of this process makes it possible to describe the reduction and multiplication functions. For unit 3, we argue in a similar way.

4. STATEMENT OF THE PROBLEM

In this section, we consider a recursive conveyor presented in the form of an acyclic connected graph for which it is necessary to minimize the execution time of \hat{k} orders by allocating resources, where \hat{k} is some constant. This RCPSP is reduced to minimizing the function $t(n, \hat{k})$ for a given \hat{k} on a finite set M of resources. Let us describe the CSP statement for discrete variables with finite sets of values.

In the theory [11], a CSP is a quadruple (V, D, R, C) , where $V = \{x_1, \dots, x_n\}$ is the set of variables, $D = \{D_1, \dots, D_n\}$ is the set of domains of variables, R is the set of relations of different valency over the domains D , and $C = \{C_1, \dots, C_m\}$ is the set of constraints binding the set of values of variables from V by means of relations in R .

To solve a CSP is to find the values of all variables in the set V satisfying all the constraints. There can be one or several CSP solutions.

The CSP can be represented as a network of constraints. For such a network it would be natural to take the graph of the conveyor model in which some operation corresponds to a vertex. However, since operation i is performed k times in the presence of k orders and some resource m_i can be used in each case, it follows that there is a certain set of resources associated with the operation. To avoid this situation, let us generate an expanded graph from the initial model graph for a given \hat{k} in which each vertex corresponds to a pair (operation number, order number) and the precedence relations are preserved. Such a graph can be constructed for any conveyor model and is not difficult.

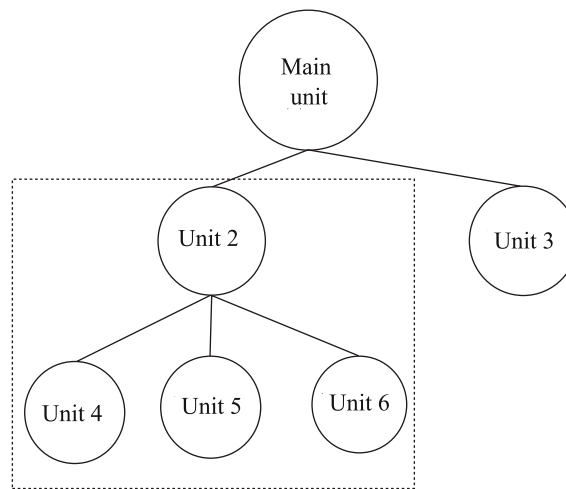


Fig. 6. Assembly item structure.

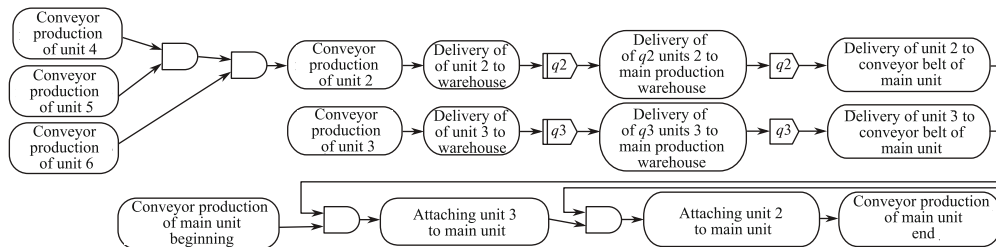
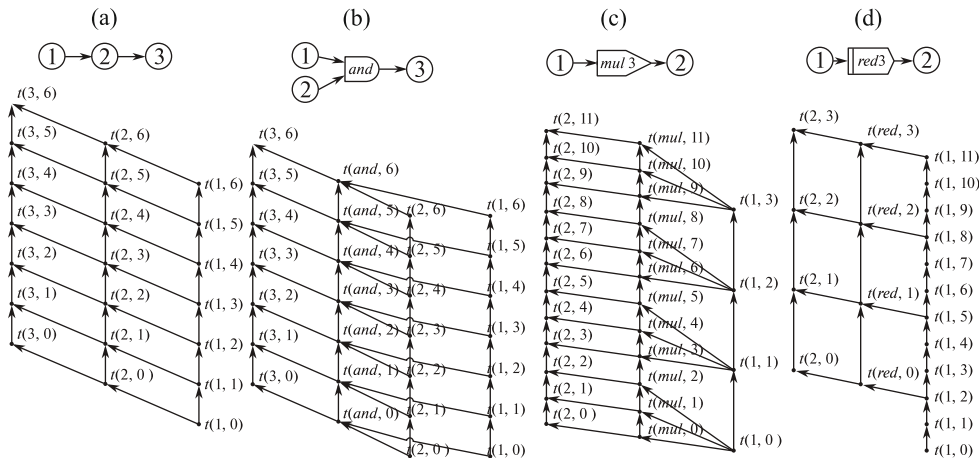


Fig. 7. Distributed assembly line model.

Fig. 8. Examples of conveyor graphs for specific values of \hat{k} .

Simple examples of such graphs are shown in Fig. 8. The top is the original graph of the model, and the bottom is the graph for a specific value of \hat{k} . The vertices of the graphs are labeled with appropriate recursive functions and are associated with specific operations and orders. If the original graph has n vertices (operations), then the new graph has $n\hat{k}$ vertices. It can readily be shown that the expanded graph is also acyclic. Obviously, in this case each vertex corresponds to one resource if the vertex corresponds to an operation (vertex type **bop** or **op**). In other cases, when the vertex corresponds to a trigger function, the resource is not used. To reflect this fact in the problem, we supplement the set of resources with the zero element, $M = \{m_0, m_1, \dots, m_r\}$.

Let us number the vertices of the expanded graph in some way. Let $I' = \{1, \dots, n'\}$, $n' = n\hat{k}$, be the set of new numbers. By default we assume that $i' \in I'$ denotes the number of a vertex in the new graph and there exist mappings $\psi : I' \rightarrow I$ and $\varphi : I' \rightarrow K$. If there is a precedence with respect to i in the original graph, then in the expanded graph there exists precedence in both i and k . Let us define these two functions. The function $pk : I' \rightarrow I'$ calculates a vertex preceding in k ; i.e., if $i'' = pk(i')$, then $\psi(i'') = \psi(i')$ and $\varphi(i'') = \varphi(i') - 1$. The function $pi : I' \rightarrow I'$ calculates a vertex preceding in i ; i.e., if $i'' = pi(i')$, then $\psi(i'') = pred(\psi(i'))$ and $\varphi(i'') = \varphi(i')$. Further, both numbering systems—consecutive with a prime and two-coordinate (i, k) without the prime—will be used in expressions to simplify the notation under the assumption that $i' \rightarrow (i, k)$, $i = \psi(i')$, and $k = \varphi(i')$.

Let us transfer the above statement of the CSP to the considered domain of scheduling theory.

The quadruple of CSP objects will be defined as follows:

1. $V = \{x_1, \dots, x_{n'}\}$ is the set of discrete variables with a range of values defined for each of them.
2. $D_{i'} = D_i$ is the domain of a variable $x_{i'}$ linked to the corresponding i th vertex of the original graph; i.e., $i' \rightarrow (i, k)$. In this case, one should bear in mind that for each vertex i' of the trigger function, the respective variables $x_{i'}$ will be defined on the domains $D_i = \{m_0\}$ containing only one zero resource. This fact substantially reduces the search space.
3. $R \subseteq (D_1 \times D_2 \times \dots \times D_{n'})$.
4. C is the set of constraints, which splits into several groups.

The set of constraints is basically a set of recursive functions derived from recursive operation functions. The modifications are caused by the transition to the expanded graph and the associated change of variables. These functions generate constraints due to the admissible schedules for each conveyor operation for a certain specified resource allocation for (operation, order) pairs. The last condition checks the admissibility of a schedule from the viewpoint of the fact that the resource is not used by two operations at some point in time.

The 1st group is the set of unary constraints defined for the initial operations ($1 \leq i \leq n_0$) and constructed based on the function (2),

$$bop(i, k) = \begin{cases} p_{ij} & \text{if } (x_{i'} = m_j) \& (k = 0) \\ bop(i, k-1) + p_{ij} & \text{if } (x_{i'} = x_{pk(i')}) \& (x_{i'} = m_j) \& (0 < k \leq \hat{k}) \\ bop(i, k-1) - p_{ij} + p_{il} & \text{if } (x_{i'} \neq x_{pk(i')}) \& (x_{i'} = m_l) \& (x_{pk(i')} = m_j) \& (0 < k \leq \hat{k}). \end{cases}$$

The 2nd group is the set of binary constraints defined for a pair of graph vertices linked by an edge and constructed based on the functions (3) and (5)–(8),

$$op(i, k) = \begin{cases} t(pred(i), k) + p_{ij} & \text{if } (type(i) = \mathbf{op}) \& (x_{i'} = m_j) \& (k = 0) \\ t(pred(i), k) + p_{ij} & \text{if } (type(i) = \mathbf{op}) \& (t(pred(i), k) \geq t(i, k-1)) \& (x_{i'} = m_j) \& (0 < k \leq \hat{k}) \\ t(pred(i), k) + p_{ij} & \text{if } (type(i) = \mathbf{op}) \& (t(pred(i), k) < t(i, k-1)) \\ & \& (x_{i'} \neq x_{pk(i')}) \& (x_{i'} = m_j) \& (0 < k \leq \hat{k}) \\ t(i, k-1) + p_{ij} & \text{if } (type(i) = \mathbf{op}) \& (t(pred(i), k) < t(i, k-1)) \\ & \& (x_{i'} = x_{pk(i')}) \& (x_{i'} = m_j) \& (0 < k \leq \hat{k}), \end{cases}$$

$$\begin{aligned}
mul(i, k) &= t(pred(i), \lfloor k/q_i \rfloor) & \text{if } (type(i) = \mathbf{mul}) \& (0 \leq k \leq \hat{k}), \\
red(i, k) &= t(pred(i), (k+1)q_i - 1) & \text{if } (type(i) = \mathbf{red}) \& (0 \leq k \leq \hat{k}), \\
get1(i, k) &= t(pred(i), 2k) & \text{if } (type(i) = \mathbf{get1}) \& (k \bmod 2 = 0) \& (0 \leq k \leq \hat{k}), \\
get2(i, k) &= t(pred(i), 2k+1) & \text{if } (type(i) = \mathbf{get2}) \& (k \bmod 2 = 1) \& (0 \leq k \leq \hat{k}).
\end{aligned}$$

The 3rd group is the set of constraints defined for a triple of graph vertices and constructed in accordance with the functions (4) and (9),

$$\begin{aligned}
and(i, k) &= \max \left(t(pred1(i), k), t(pred2(i), k) \right) \text{ if } (type(i) = \mathbf{and}), \\
put(i, k) &= \begin{cases} t(pred1(i), k) & \text{if } (type(i) = \mathbf{put}) \& (k = 0) \\ \max \left(put(i, k-1), t(pred2(i), k/2) \right) & \text{if } (type(i) = \mathbf{put}) \\ & \& (k \bmod 2 = 0) \& (0 < k \leq \hat{k}) \\ \max \left(put(i, k-1), t(pred1(i), (k-1)/2) \right) & \text{if } (type(i) = \mathbf{put}) \\ & \& (k \bmod 2 = 1) \& (0 < k \leq \hat{k}). \end{cases}
\end{aligned}$$

The 4th group of constraints refers to constraints like **all-different**. It reflects the fact that the use of a resource at a time is possible by no more than one operation. Let us introduce the following notation: $\tau_{i'} = (t_{i'}^b, t_{i'}^e)$ denotes the time interval from $t_{i'}^b$ to $t_{i'}^e$ ($t_{i'}^b \leq t_{i'}^e$) for which the i th operation executes the k th order ($i = \psi(i')$, $k = \varphi(i')$) with some resource m_j used. The operation \cap takes the value “True” if the time intervals overlap,

$$\tau_{i'} \cap \tau_{i''} = \begin{cases} \text{True} & \text{if } (t_{i'}^b < t_{i''}^e \leq t_{i'}^e) \text{ or } (t_{i'}^b \leq t_{i''}^b < t_{i'}^e) \\ \text{False} & \text{otherwise.} \end{cases}$$

With these definitions, the global constraint is as follows:

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

The meaning of the condition is that if the intervals of two distinct vertices i' and i'' overlap, then either at least one of them corresponds to a trigger function or they use distinct resources.

To solve the CSP is to find the values of the variables $x_{i'}$ for which all the listed conditions are satisfied; it is obvious that in this statement the CSP will always have some solution. One can compile a schedule for any allowed resource allocation. The present paper requires an optimal schedule. Let T_{opt} be the execution time of the optimal schedule. We will assume that the solution is quasioptimal up to some predetermined value Δ if the solution T found is such that $T - T_{\text{opt}} \leq \Delta$. The calculation of the quasioptimal value of the schedule execution time can be carried out using the following well-known algorithm.

Algorithm.

Step 1.

Choose some set $V = V_0$ of values.

Calculate the value of $T_{\text{max}} = t(n', \hat{k})$ for V_0 .

Set $T_{\text{min}} = 0$.

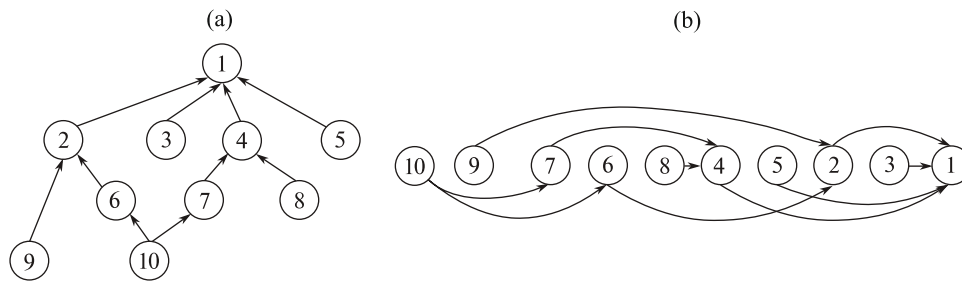


Fig. 9. Example of a linear order on a graph.

Step 2. If $T_{\max} - T_{\min} \leq \Delta$, then the problem has been solved, T_{\max} is a solution, and the set V corresponding to this solution is the desired resource distribution.

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

Step 4. Introduce the auxiliary constraint $t(n', \hat{k}) \leq T$ in the CSP.

Step 5. If the CSP with the auxiliary condition does not have a solution, then set $T_{\min} = T$ and go to step 3.

Step 6. If the CSP with the auxiliary condition has a solution, then set $T_{\max} = T$ and go to step 2.

The value T_{\max} thus found is quasiminimal, but the quasioptimality of this value needs to be proved. The essence of the problem is that recursive functions construct an optimal schedule at the local level (implement a greedy algorithm), and it must be proved that the local optimization is also global for a fixed resource allocation.

Assertion. For each feasible value $V = \{x_1, \dots, x_{n'}\}$, the recursive function $t(i, k)$ calculates one of the optimal values for $1 \leq i \leq n$ and $0 \leq k \leq \hat{k}$.

Proof. We prove the assertion by induction.

For the presentation to be systematic, we recall the statement of the induction principle. Some assertion $P(i)$ depending on a positive integer i is considered proven if

The base case. It is established that $P(1)$ holds true.

The induction step. For each $1 \leq i < n'$, the assumption that $P(i)$ holds implies that $P(i+1)$ holds true as well.

Let us apply this method to the expanded graph. It is well known from graph theory [17] that an acyclic graph is strictly partially ordered. There are algorithms for the construction of a linear order on a strictly partially ordered graph. Informally, we speak of a linear order if all the vertices of the graph are arranged in a row and all edges are directed from left to right. An example of constructing a linear order from a strictly partial order is shown in Fig. 9, where panel (a) is the original acyclic graph and panel (b) is one of the possible linear orders. In this paper, any such algorithm will be suitable provided that the first n_0 vertices of the original graph still remain the first, albeit possibly in a different order, and the vertex n remains the last one. When evaluating a recursive function in accordance with a linear order, its arguments will have already been calculated.

We apply the induction method as follows.

The base case. Based on the definition of a recursive conveyor and the way of ordering the vertices in the original graph, it is obvious that all the initial vertices in the original graph are operations (not functions) and are derivatives for the initial vertices of the expanded graph. If the initial graph has n_0 initial vertices, then in the expanded graph there are also n_0 initial vertices, and for each initial vertex i in the original graph there is a vertex $(0, i)$ in the expanded graph that

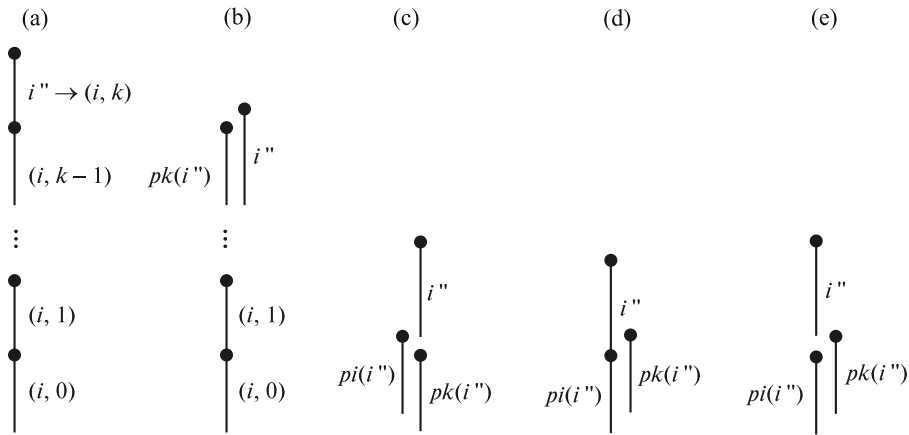


Fig. 10. Variants of fragments of timing diagrams.

is not preceded by any other vertex. If $1 \leq i \leq n_0$ is some initial vertex of the original graph, then the vertex i' such that $\psi(i') = i$ and $\varphi(i') = 0$ is the starting vertex of the expanded graph.

Thus, for each initial vertex $i' \rightarrow (i, k)$ of the expanded graph the following assertions hold:

$$\text{type}(i) = \mathbf{bop}, \quad 1 \leq \psi(i') \leq n_0 \text{ and } \varphi(i') = 0, \text{ i.e., } (1 \leq i \leq n_0) \& (k = 0).$$

In this case, in accordance with function (1), we have

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

Thus, for a particular value $x_{i'}$, the value $t(i, 0)$ is unique and hence optimal. The Assertion holds true. Since this assertion holds for any of the n_0 initial operations, we will not consider these variants below.

The induction step. Suppose that the Assertion holds for a vertex with any number $n_0 < i' < n'$. Let us prove that in this case it also holds for the vertex with number $i'' = i' + 1$. Eight cases are possible in accordance with (1).

Consider cases 1 and 2, when

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

Case 1. The initial operation is performed over a nonzero order; i.e., $i'' \rightarrow (i, k)$ and $(1 \leq i \leq n_0) \& (k > 0)$. If $x_{i''} = x_{pk(i'')}$, then operation i , subject to the completion of the k th and $(k - 1)$ st orders, uses one resource.

In this case, the completion time is calculated by formula (2(a)) (the diagram in Fig. 10a). If distinct resources are used, then operation i can be performed simultaneously on orders $(k - 1)$ and k . The completion time is calculated by formula (2(b)) (the diagram in Fig. 10(b)). The duration of the operations may not be the same. Considering that the execution time of the $(k - 1)$ st order by operation i is optimal, the execution time of the k th order by i is optimal as well. This completes the analysis of this case.

Case 2. A noninitial operation is performed in this case. If $k = 0$, then the completion time is calculated by formula (3(a)). If $k > 0$, then the completion time is calculated based on cases (3(b))–(3(d)). The corresponding examples of timing diagrams are shown in Figs. 10c–10e. In all cases, the new schedule with the addition of vertex i'' will be optimal if the previous schedule was optimal.

Cases 3–8. In all these cases, the vertices correspond to trigger functions. Hence the resources are not allocated and the final schedule is constructed in the only possible way based on the temporal precedence relation determined by the corresponding recursive function.

This completes the proof of the Assertion. ■

The time complexity of the Algorithm is

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

where T_0 is the value of the execution time of the schedule for some arbitrary set of values V_0 , Δ is some predetermined constant of the precision of the calculation of the result, and $|D_i| = 1$ for trigger functions.

5. CONCLUSIONS

The present paper gives a theoretical statement of the problem. For the first time, a tool such as Constraint Programming is used to study the conveyor model. Applying this method in practice is a further direction of research of the present authors.

There are a large number of commercial and freeware *constraint programming* systems nowadays. An overview of such systems can be found in [2].

The study of the computational complexity of solving the CSP seems to be a fundamental problem. It was shown in [18] that the class of all CSPs is NP-hard, so there are hardly any efficient (polynomial) general-purpose algorithms for solving all types of CSPs. Classes of easily solvable CSPs that can be solved in polynomial time are usually described either by trees and tree-like graph structures [19, 20] or by a certain combination of algebraic operators [21]. The problem considered in the paper is based on a tree of constraints (the original acyclic graph and the extended graph); this inspires optimism about the existence of an efficient algorithm for solving the problem. Constructing a schedule for a conveyor by calculating the superposition of recursive functions assumes the *backtracking* option in various modifications [2] for solving the CSP. It is important to note that the introduction of trigger functions into the conveyor model does not in any way increase the algorithmic complexity of solving the problem.

There are two important advantages of this method.

First, the recursive conveyor optimization method discussed in the paper [22] is reduced to an integer linear programming problem and considers a strictly defined set of recursive functions. The introduction of new functions requires additional consideration of the correctness of the application of the method. The approach used in this paper, although it considers a specific set of functions, does not impose any restrictions on these functions other than computability. This allows introducing new recursive functions into consideration without discussing the issue of the correctness of the method application.

Second, the proposed method allows one to distribute resources without assigning them to separate operations. This approach increases the dimension of the problem but opens up opportunities for constructing more efficient schedules.

FUNDING

This work was supported by the Russian Foundation for Basic Research, project no. 20-58-S52006.

REFERENCES

1. Lazarev, A.A. and Gafarov, E.R., *Teoriya raspisaniy. Zadachi i algoritmy* (Scheduling Theory. Problems and Algorithms), Moscow: Izd. Mosk. Gos. Univ., 2011.

2. Rekiek, B., Dolgui, A., Delchambre, A., and Bratcu, A., State of art of optimization methods for assembly line design, *Annu. Rev. Control*, 2002, vol. 26, pp. 163–174.
3. Ghosh, S. and Gagnon, R.J., A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems, *Int. J. Product. Res.*, 1989, vol. 26, no. 4, pp. 637–6670.
4. Helgeson, W.B., Salvesson, M.E., and 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), pp. 3850–389.
6. Salvesson, M.E., The assembly line balancing problem, *J. Ind. Eng.*, 1955, no. 6, pp. 18–25.
7. Neumann, K., Schwindt, C., and Zimmermann, J., Recent results on resource-constrained project scheduling with time windows: Models, solution methods, and applications, *Cent. Eur. J. Oper. Res.*, 2002, vol. 10, no. 2, pp. 113–148.
8. Drexel, A. and Kimms, A., Optimization guided lower and upper bounds for the resource investment problem, *J. Oper. Res. Soc.*, 2001, vol. 52, no. 3, pp. 340–351.
9. Ranjbar, M., Kianfar, F., and Shadrokh, S., Solving the resource availability cost problem in project scheduling by path relinking and genetic algorithm, *Appl. Math. Comput.*, 2008, vol. 196, no. 2, pp. 879–888.
10. Yamashita, D.S., Armentano, V.A., and Laguna, M., Robust optimization models for project scheduling with resource availability cost, *J. Sched.*, 2007, vol. 10, no. 1, pp. 67–76.
11. Shcherbina, O.A., Constraint satisfaction and constraint programming, *Intell. Syst.*, 2011, vol. 15, no. 1–4, pp. 53–170.
12. Apt, K.R., *Principles of Constraint Programming*, New York: Cambridge Univ. Press, 2003.
13. Narin'yan, A.S., Sedreeva, G.O., Sedreev, S.V., and Frolov, S.A., TimeEX/Windows—a new generation of underdetermined scheduling technology, in *Problemy predstavleniya i obrabotki ne polnost'yu opredelennykh znaniy* (Problems of Representation and Processing of Incompletely Defined Knowledge), Shvetsov, I.E., Ed., Moskva–Novosibirsk, 1996, pp. 101–116.
14. Kupriyanov, B.V., Recursive conveyor processes—basic properties and characteristics, *Ekonom. Stat. Inf. Vestn. UMO*, 2015, no. 1, pp. 163–170.
15. Kupriyanov, B.V., Application of a model of conveyor processes of recursive type for solving applied problems, *Ekonom. Stat. Inf. Vestnik UMO*, 2014, no. 5, pp. 170–179.
16. Kupriyanov, B.V., Method of Efficient Analysis of the Recursive Conveyor Process Models, *Autom. Remote Control*, 2017, vol. 78, no. 3, pp. 435–449.
17. Ore, O., *Theory of Graphs*, Providence, Rhode Island: Am. Math. Soc., 1962. Translated under the title: *Teoriya grafov*, Moscow: Nauka, 1980.
18. Mackworth, A.K., Consistency in networks of relations, *Artif. Intell.*, 1977, vol. 8, no. 1, pp. 99–118.
19. Dechter, R. and Pearl, J., Tree clustering for constraint networks, *Artif. Intell.*, 1989, vol. 38, no. 3, pp. 353–366.
20. Freuder, E.C., Backtrack-free and backtrack-bounded search, in *Search in Artificial Intelligence*, Kanal, L. and Kumar, V., Eds., Heidelberg: Springer-Verlag, 1988, pp. 343–369.
21. Jeavons, P.G., Cohen, D.A., and Gyssens, M., Closure properties of constraints, *J. ACM*, 1997, vol. 44, pp. 527–548.
22. Kupriyanov, B.V., Estimation and optimization of the efficiency of a recursive conveyor, *Autom. Remote Control*, 2020, vol. 81, pp. 775–790.

This paper was recommended for publication by A.A. Galyaev, a member of the Editorial Board