On the Recursive Representation of the Permutation Flow Shop Scheduling Problem and Its Extension

Boris Kupriyanov , Frank Werner * , Alexander Lazarev , Alexander Roschin

Posted Date: 27 November 2024

doi: 10.20944/preprints202411.2070.v1

Keywords: flow shop; recursive functions; branch and bound method; scheduling

Article

On the Recursive Representation of the Permutation Flow Shop Scheduling Problem and Its Extension

Boris Kupriyanov [1], Alexander Lazarev [1,†], Alexander Roschin [1] and Frank Werner [2,*]

[1] Institute of Control Sciences, 65 Profsoyuznaya street, 117997 Moscow, Russia; kuprianovb@mail.ru (B.K.); jobmath@mail.ru (A.L.); rochinaa@ipu.ru (A.R.)
[2] Fakultät für Mathematik, Otto-von-Guericke-Universität Magdeburg, PSF 4120, 39016 Magdeburg, Germany
* Correspondence: frank.werner@ovgu.de
† This work has been done during the stay of Alexander Lazarev at the Otto-von-Guericke University Magdeburg, supported by DAAD (grant №91696586).

Abstract: In this paper, we propose a formulation of the permutation flow shop scheduling problem (PFSP) by special recursive functions and show its equivalence to the existing classical formulation. Equivalence is understood in the sense that both ways of defining the problem describe the same set of feasible schedules for each pair of job and machine numbers. In fact, this paper attempts to use the apparatus of recursive functions for scheduling problems. Moreover, the predicate *and* is introduced into the problem, and the problem PFS is expanded from a chain of machines to an acyclic graph. To construct an optimal schedule, the branch and bound method is considered based on the recursive function, which is included into the corresponding job permutation algorithm. The complexity of the optimization algorithm does not increase compared to the non-recursive PFS problem. Finally, we present some initial computational results.

Keywords: flow shop; recursive functions; branch and bound method; scheduling

MSC: 90B35; 90C57

1. Introduction

There exist a huge number of publications devoted to the permutation flow shop problem (PFSP) or to the more general case of the flow shop problem (FSP) in the scheduling literature. In both the PFSP and the FSP, all jobs must be processed on set of m machines according to a fixed technological order. In the PFSP, each of the n jobs must be processed in the same sequence, while in the FSP, different job sequences can be selected to satisfy a specific optimization objective function. Often the makespan has to be minimized, i.e., the maximum completion time of a full set of jobs should be minimized. The PFSP is denoted as $F|prmu|C_{max}$. A corresponding review can be found e.g. in [1]. Most works deal with various heuristics and metaheuristics, see e.g. [2–5].

This article discusses a new approach through the definition of recursive functions to describe the timing characteristics of the PSFP. The concept of interval calculations is used to describe time domains. The equivalence of the PFSP representation in the existing and the recursive form is proved. Here, equivalence is understood as the equality of the sets of feasible schedules for each pair of job number and machine number. Then the transition from interval recursive functions to scalar ones is carried out. It is shown that with a fixed order of operations, scalar functions calculate an optimal schedule on a set of feasible schedules.

Moreover, the recursive representation of the PFSP is extended by including the predicate *and*, which allows one to move from chains of machines to trees and acyclic graphs. The branch and bound method (B&B) [6] using the introduced recursive function is considered as an exact optimization method. A description of one of the variants of a lower bound on the makespan and the branching method are given. The branching method is based on an original permutation generator with a given property. The permutation algorithm and the recursive implementation of the B&B method are

 doi:10.20944/preprints202411.2070.v1

described. It is shown that the complexity of the algorithm does not increase compared to the known B&B algorithms for the PFSP [6,10].

The rest of this paper is organized as follows. Section 2 reviews some basics of interval calculations relevant for the PFSP. Section 3 proves the equivalence of the existing and the recursive PFSP representations. Section 4 provides a method for implementing the B&B algorithm for the recursive PFSP. Section 5 gives the definition of the *and* function and the extended formulation of the problem. Then Section 6 discusses the B&B method for an extended formulation of the problem. Section 7 presents some initial results with the suggested algorithm.

2. Interval Values and Operations

Interval calculations in mathematics have been known for a long time, see for example [7]. In our case, this theory is used at the level of definitions and properties of some interval operations. We will consider the set of values of the time interval as a segment of integers on the number line with starting and end points included.

Definition 1. *Let $N = \{0, 1, 2, 3, \dots, \hat{N}\}$ be a finite set of natural numbers from one to $\hat{N}$ with zero added. Everywhere below, by an interval or domain $[a, b]$, $a \le b$, unless otherwise stated, we denote the closed bounded subset N of the form*

$$[a, b] = \{x |\ x \in N \wedge a \le x \le b\}. \tag{1}$$

The set of all intervals is denoted by $\mathcal{I}$. We will write the elements of $\mathcal{I}$ in capital letters. If $A \in \mathcal{I}$, then we will denote its left and right end points as $\underline{a}$ and $\overline{a}$: $A = [\underline{a}, \overline{a}]$. We will call the elements of $\mathcal{I}$ interval numbers. Two intervals A and B are equal if and only if $\underline{a} = \underline{b}$ and $\overline{a} = \overline{b}$. According to the total interval approach [7], we can describe operations as follows. Let $\circ$ be some operation and $A, B \in \mathcal{I}$. In this case, we have

$$A \circ B = \{a \circ b |\ a \in A,\ b \in B\}.$$

If $A = [\underline{a}, \overline{a}]$ and $B = [\underline{b}, \overline{b}]$, then $A \circ B = [\underline{a} \circ \underline{b}, \overline{a} \circ \overline{b}]$. We will use only two interval operations: "+" and "max". They are defined as follows:

$$\begin{aligned} A + B &= A = [\underline{a} + \underline{b}, \overline{a} + \overline{b}]; \\ \max\{A, B\} &= [\max\{\underline{a}, \underline{b}\}, \max\{\overline{a}, \overline{b}\}]. \end{aligned}$$

To illustrate, consider the following example.

Example 1. *Let $A = [12, 17]$ and $B = [3, 5]$. Then*

$$\begin{aligned} A + B &= [12, 17] + [3, 5] = [12 + 3, 17 + 5] = [15, 22]; \\ \max\{A, B\} &= \max\{[12, 17], [3, 5]\} = [\max\{12, 3\}, \max\{17, 5\}] \\ &= [12, 17]. \end{aligned}$$

If $A = [6, 9]$ and $B = [3, 12]$, then

$$\max\{A, B\} = \max\{[6, 9], [3, 12]\} = [\max\{6, 3\}, \max\{9, 12\}] = [6, 12].$$

While in the first case, the interval A is obtained by the operation max, *in the second case a subinterval of B results.*

To illustrate, below are some further examples of various options for interval intersection.
$\max\{[1, 4], [2, 3]\} = [2, 4]$;
$\max\{[2, 5], [3, 6]\} = [3, 6]$;
$\max\{[5, 15], [0, 0]\} = [5, 15]$.

 doi:10.20944/preprints202411.2070.v1

A confluent interval, i.e., an interval with coinciding end points $a = \underline{a} = \overline{a}$, is identical to the integer a. Thus, $N \subset \mathcal{I}$. If A and B are confluent intervals, then the results of the operations on intervals coincide with ordinary arithmetic operations for positive integers. An interval integer is a generalization of an integer, and the interval integer arithmetic is a generalization of the integer arithmetic.

Note also that for $B = [b, b]$, we write for simplicity also $A + B = A + b$. In the theory of intervals, the role of zero is played by the usual 0, which is identified with the confluent interval $[0, 0]$. In other words,

$$A + 0 = 0 + A = A, \ \max\{A, 0\} = \max\{0, A\} = A, \ \forall A \in \mathcal{I}.$$

Calculations can lead to an interval $\emptyset = [a, b]$ if $a > b$, which, according to Definition 1, represents an empty set.

3. Solving the Permutation Flow Shop Equivalence Problem

Consider the description of the PFSP [8]. Let $\mathcal{J} = \{1, 2, \ldots, n\}$ be the set of jobs and $\mathcal{M} = \{1, 2, \ldots, m\}$ be the set of machines arranged sequentially. Moreover, let $p_{j,k}$ be the processing time of job j on machine k.

The precedence graph of such a problem (see Figure 1a) is a simple chain in which the vertices are machines, and the edges define precedence relations between the machines. This graph can be "expanded" if we fix some order of the jobs and define precedence relations between the jobs $\mathcal{J} \times \mathcal{J}$ and between the machines $\mathcal{M} \times \mathcal{M}$. Any vertex of such a graph is identified with the pair (job number, machine number). An expanded graph describing precedence relations between the jobs is presented in Figure 1b.

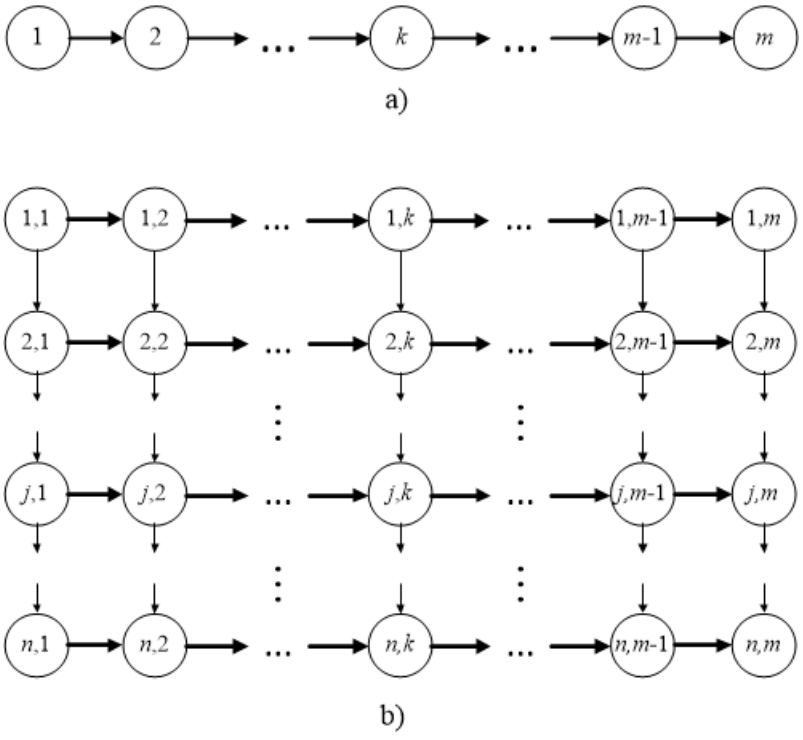

Figure 1. Example of a permutation flow shop problem graph

The vertical edges define precedence relations between the jobs for a particular machine. The horizontal edges define precedence relations between the machines for a fixed job. This graph is acyclic and has one start and one end vertex. Consider some schedule and then renumber the jobs in the order

 doi:10.20944/preprints202411.2070.v1

in which they are located in the schedule, i.e., for simplicity of the description consider the schedule $(1, 2, ..., n)$ with the new numbering. The following assumptions are valid for problem PFSP:

1. Job assumptions: A job cannot be processed by more than one machine at a time; each job must be processed in accordance with the machine precedence relationship; the job is processed as early as possible, depending on the order of the machines; all jobs are equally important, meaning there are no priorities, deadlines or urgent orders.
2. Machine assumptions: no machine can process more than one job at a time; after starting a job, it must be completed without interruption; there is only one machine of each type; no job is processed more than once by any machine.
3. Assumptions about the processing times: the processing time of each job on each machine does not depend on the sequence in which the jobs are processed; the processing time of each job on each machine is specific and integer; the transport time between machines and a setup time, if any, are included in the processing time.

Additional time limits can be set in the scheduling problems. Denote $T(j,k) = [\underline{t}_{j,k}, \bar{t}_{j,k}]$ as the interval completion time of job j by the kth machine. $\underline{t}_{j,k}$ is the minimal time for the completion of job j on the kth machine, and $\bar{t}_{jk}$ is the corresponding maximal time. The completion time $t_{j,k}$ must satisfy the condition $\underline{t}_{j,k} \le t_{j,k} \le \bar{t}_{j,k}$, otherwise the schedule is infeasible. In fact, this is the interval function completion times of job j on machine k. We will assume that in the initial formulation of the problem, a matrix $\|T^0(j,k)\| = \|[\underline{t}^0_{j,k}, \bar{t}^0_{j,k}]\|$ of initial domains of the feasible values for the job completion times has been defined. We will assume, unless otherwise specified, that the initial domains are the interval $[0, T_{max}]$, where T_{max} is a sufficiently large finite value which will be discussed below. We describe two additional restrictions [8]:

- r_j is the moment when job j arrives for processing, i.e., its release date. This parameter defines the point in time from which job j can be scheduled for execution, but its processing does not necessarily begin at this moment. In this case, in the matrix of initial domains, it is necessary to put $T^0(j,k) = [r_j + p_{j,k}, T_{max}]$ for $j \in \mathcal{J}$ and $1 \le k \le m$.
- D_j is the deadline for processing job j. A deadline cannot be violated, and any schedule that contains a job that finishes after its deadline is infeasible. In this case, in the matrix of initial domains, it is necessary to put $T^0(j,k) = [0, D_j]$ for $j \in \mathcal{J}$ and $1 \le k \le m$.

The combination of the constraints r_j and D_j generates initial domains of the form $T^0(j,k) = [r_j + p_{j,k}, D_j]$ for $j \in \mathcal{J}$ and $1 \le k \le m$. If these restrictions relate to specific operations, they will look like $r_{j,k}$ and $D_{j,k}$. In this case, the following time relations will be satisfied:

$$p_{1,1} \le t_{1,1}, \;\; t_{1,1} \le t_{1,2} - p_{1,2}, \; \ldots, \; t_{1,m-1} \le t_{1,m} - p_{1,m}. \tag{2}$$

$$t_{1,1} \le t_{2,1} - p_{2,1}, \;\; \max(t_{1,2}, t_{2,1}) \le t_{2,2} - p_{2,2}, \; \ldots, \; \max(t_{1,m}, t_{2,m-1}) \le t_{2,m} - p_{2,m}. \tag{3}$$

$$\vdots$$

$$t_{n-2,1} \le t_{n-1,1} - p_{n-1,1}, \;\; \max(t_{n-2,2}, t_{n-1,1}) \le t_{n-1,2} - p_{n-1,2}, \; \ldots, \; \max(t_{n-2,m}, t_{n-1,m-1}) \le t_{n-1,m} - p_{n-1,m}. \tag{4}$$

$$t_{n-1,1} \le t_{n,1} - p_{n,1}, \;\; \max(t_{n-1,2}, t_{n,1}) \le t_{n,2} - p_{n,2}, \; \ldots, \; \max(t_{n-1,m}, t_{n,m-1}) \le t_{n,m} - p_{n,m}. \tag{5}$$

Here we take into account the fact that the interval time $B(j,k)$ for the start of job j on the kth machine is calculated by the formula

$$B(j,k) = [\underline{t}_{j,k}, \bar{t}_{j,k}] - p_{j,k} = [\underline{t}_{j,k} - p_{j,k}, \bar{t}_{j,k} - p_{j,k}], \;\; 1 \le j \le n, \; 1 \le k \le m.$$

The above inequalities are supplemented by inequalities of the form $\underline{t}_{j,k} \le \bar{t}_{j,k}$.

 doi:10.20944/preprints202411.2070.v1

The system of inequalities (2)-(5) defines an infinite set of feasible schedules. To reduce this set to a finite one, we will use a positive integer constant T_{max} and supplement the system (2)-(5) with the inequalities

$$\bar{t}_{j,k} \leq T_{max}, \quad 1 \leq j \leq n, 1 \leq k \leq m. \tag{6}$$

It is possible to assign to T_{max} a value so small that the set of feasible schedules is empty. We assume that the value of T_{max} is such that there is at least one feasible schedule.

Example 2. *Consider a PFSP with 3 machines and 4 jobs.*

Table 1 shows the processing times $p_{j,k}$ of the jobs. The rows correspond to the job numbers, and the columns represent the machine numbers.

Table 1. Table of the job processing times.

Job\Machine	1	2	3
1	3	2	2
2	3	1	3
3	3	2	2
4	3	1	2

Let $T_{max} = 16$. In this case, the intervals for each pair (j,k) are given in Table 2.

Table 2. Table of the job completion time intervals.

Job\Machine	1	2	3
1	[3,4]	[5,7]	[7,9]
2	[6,7]	[7,9]	[10,12]
3	[9,10]	[11,12]	[13,14]
4	[12,13]	[13,14]	[15,16]

Figure 2 shows the Gantt charts of three variants from the set of feasible schedules. "Schedule 1" and "Schedule 2" are optimal schedules (for this sequence), while "Schedule 2" and "Schedule 3" have different inserted idle times. It is easy to verify that increasing any interval in Table 2 will lead to the appearance of infeasible schedules, and decreasing any interval will result in the loss of feasible schedules.

 doi:10.20944/preprints202411.2070.v1

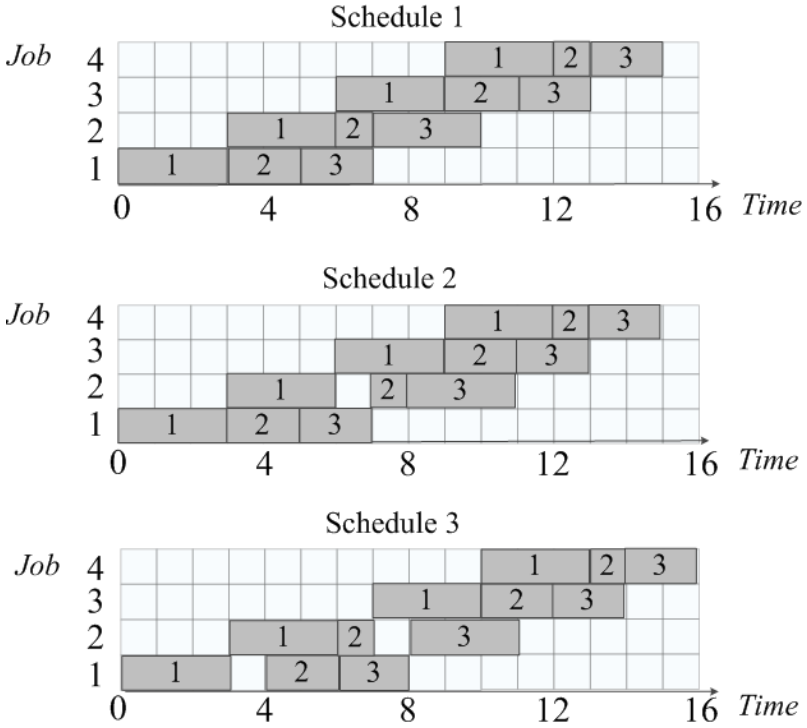

Figure 2. Examples of diagrams of feasible schedules

Let $T'(j,k)$ be the interval calculated by some algorithm for completing the jth job on the kth machine and $T^0(j,k)$ be the corresponding initial domain. According to Figure 3, we define the interval noncommutative operation intersection ($\cap$) as $T(j,k) = T'(j,k) \cap T^0(j,k)$. The figure shows all significant cases of interval intersections:

a) the interval $T'(j,k)$ is to the left of $T^0(j,k)$. The completion time of the jth job on the kth machine should be artificially increased to values from the interval $T^0(j,k)$. Only in this case, the schedule wil be feasible;
b-e) are cear without additional explanation;
f) with this arrangement of intervals, the value of the interval $T(j,k)$ is equal to an empty set, which corresponds to the absence of a feasible schedule.

Thus, we have:

$$T(j,k) = \begin{cases} \varnothing, & \overline{T}^0(j,k) < \underline{T}'(j,k); \\ \left[\max\{\underline{T}'(j,k), \underline{T}^0(j,k)\}, \overline{T}^0(j,k)\right], & \textit{otherwise}. \end{cases} \tag{7}$$

 doi:10.20944/preprints202411.2070.v1

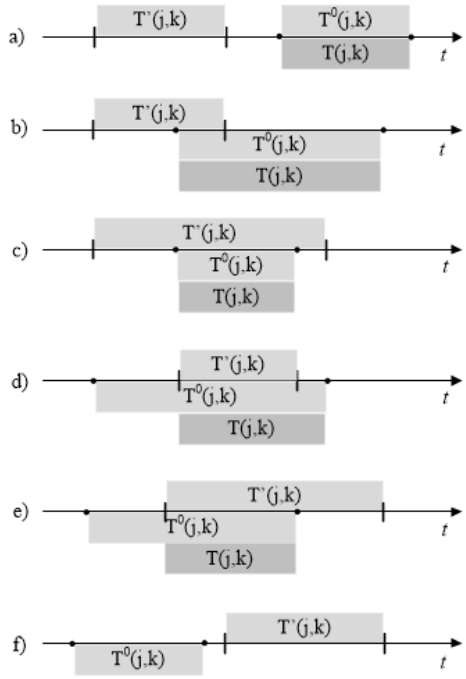

Figure 3. Cases of interval intersections

It can be shown that all feasible sets of time values are uniquely determinable and maximal. Therefore, when searching for an optimal solution, no solution will be lost.

All precedence relations define intervals (including the scalar $c = [c, c]$). Again for simplicity of the description, consider some schedule and then renumber the jobs in the order in which they are located in the schedule, i.e., consider then the schedule $(1, 2, ..., n)$ with the new numbering. The calculations of these intervals can be divided into 4 groups. Next, we write them using interval variables, assuming that the initial interval is $T^0(j,k)$ and $\overline{T}^0(j,k) \leq T_{max}$:

1. For the completion of the 1st job on the 1st machine $(j = 1,\ k = 1)$, we have:

$$\begin{aligned} T'(1,1) &= [p_{1,1}, p_{1,1}], \\ T(1,1) &= T'(1,1) \cap T^0(1,1). \end{aligned}$$

2. For the completion of the 1st job on the kth machine $(j = 1,\ 1 < k \leq m)$, we have:

$$\begin{aligned} T'(1,k) &= T(1,k-1) + [p_{1,k}, p_{1,k}], \\ T(1,k) &= T'(1,k) \cap T^0(1,k). \end{aligned}$$

3. For the completion of the job j on the 1st machine $(1 < j \leq n,\ k = 1)$, we have:

$$\begin{aligned} T'(j,1) &= T(j-1,1) + [p_{j,1}, p_{j,1}], \\ T(j,1) &= T'(j,1) \cap T^0(j,1). \end{aligned}$$

4. The interval of the completion time of job j on the kth machine $(1 < j \leq n,\ 1 < k \leq m)$ is obtained as follows:

$$\begin{aligned} T'(j,k) &= \max\{T(j-1,k), T(j,k-1)\} + [p_{j,k}, p_{j,k}], \\ T(j,k) &= T'(j,k) \cap T^0(j,k). \end{aligned}$$

The PFSP can be formulated using a finite set of special recursive functions. This is due to the existence of precedence relationships between the machines and jobs in the PFSP when the current characteristics

 doi:10.20944/preprints202411.2070.v1

of the problem are determined by the preceding ones. Recursive functions have *arguments* and *parameters* as data. The arguments of the functions are pairs (job number, machine number). They are passed as function arguments in the normal mathematical sense. Parameters are defined outside the recursive function, and they are essentially *global variables* in imperative programming terms. Examples of parameters can be: n, m, T_{max}, $\|p_{j,k}\|$, $\|T^0(j,k)\|$, and so on. The calculated value (or set of values) of the function is the completion time of job j on the kth machine.

We will denote the interval recursive function as $\mathcal{T} : \mathcal{J} \times \mathcal{M} \to \mathcal{I}$. Specific types of recursive functions will be provided subsequently as the PFSP is considered. Let us consider the problem of equivalence of the existing PFSP formulation in the literature and its recursive representation. The equivalence of both formulations is established through the equivalence of the domains of acceptable values of the completion times for each pairs (job number, machine number). An optimal solution is within the allowed processing times. This fact allows us to move to the functional representation of the problem and to apply various appropriate methods. We can describe each group by the corresponding recursive functions:

1. $\mathcal{T}(1,1) = [p_{1,1}, p_{1,1}] \cap T^0(1,1)$; $j = 1, k = 1$.
2. $\mathcal{T}(1,k) = (\mathcal{T}(1,k-1) + [p_{1,k}, p_{1,k}]) \cap T^0(1,k)$; $j = 1, 1 < k \leq m$.
3. $\mathcal{T}(j,1) = (\mathcal{T}(j-1,1) + [p_{j,1}, p_{j,1}]) \cap T^0(j,1)$; $1 < j \leq n, k = 1$.
4. $\mathcal{T}(j,k) = (\max\{\mathcal{T}(j-1,k), \mathcal{T}(j,k-1)\} + [p_{j,k}, p_{j,k}]) \cap T^0(j,k)$; $1 < j \leq n, 1 < k \leq m$.

This set of functions can be combined into one function:

$$\mathcal{T}(j,k) = \begin{cases} [p_{1,1}, p_{1,1}] \cap T^0(1,1), & j = 1, k = 1; \\ (\mathcal{T}(1,k-1) + [p_{1,k}, p_{1,k}]) \cap T^0(1,k), & j = 1, 1 < k \leq m; \\ (\mathcal{T}(j-1,1) + [p_{j,1}, p_{j,1}]) \cap T^0(j,1), & 1 < j \leq n, k = 1; \\ (\max\{\mathcal{T}(j-1,k), \mathcal{T}(j,k-1)\} + [p_{j,k}, p_{j,k}]) \cap T^0(j,k), & 1 < j \leq n, 1 < k \leq m. \end{cases} \tag{8}$$

The function $\mathcal{T}(j,k)$ is defined for all values of j and k. If $\mathcal{T}(j,k) = \emptyset$, then there is no feasible schedule. It can be seen from the formula that it calculates the only value of the interval. We show that this recursive function defines the same domain for each (j,k) as the set of values defined by the system of inequalities (2)-(6).

Theorem 1. *Let the following conditions be satisfied for both the PFSP given by the system of inequalities (2)-(6) and the recursive PFSP:*

- *one graph is set for both problems (see Figure 1a);*
- *consider some schedule and then renumber the jobs in the order in which they are located in the schedule, i.e., consider the schedule $(1, 2, ..., n)$ with the new numbering;*
- *the processing time of job j on the kth machine is $p_{j,k}$;*
- *the same matrix of initial time intervals is specified:*

$$\|T^0(j,k)\| = \|[\underline{t}^0_{j,k}, \overline{t}^0_{j,k}]\|.$$

Then the following statement is true: the set of domains (domain of definition) $\mathcal{D}(j,k)$ of feasible times for the non-recursive PFSP coincides with the set of domains $\mathcal{D}'(j,k)$ for the recursive PFSP, i.e., $\mathcal{D}(j,k) = \mathcal{D}'(j,k)$ for $1 \leq j \leq n$, $1 \leq k \leq m$.

In other words, the sets of valid schedules for each pair (job number, machine number) are the same, or in both cases there is no feasible schedule, i.e., there are j and k such that $\mathcal{D}_{j,k} = \mathcal{D}'_{j,k} = \emptyset$.

Proof. Let the conditions of the theorem be satisfied. The proof is done by complete induction. Let us recall the formulation of the principle of complete induction. Let there be a sequence of statements

 doi:10.20944/preprints202411.2070.v1

$P_1, P_2, P_3, \ldots$. If for any natural i the fact that all $P_1, P_2, P_3, \ldots, P_{i-1}$ are true also implies that P_i is true, then all statements in this sequence are true. So for $i \in N, i \geq 2$ we have

$$(\forall j \in \{1; \ldots; i-1\} P_j \Rightarrow P_i) \Rightarrow (\forall i \in N) P_i.$$

Here $x \Rightarrow y$ denotes a logical implication. The statement P_i is the equality of the intervals $\mathcal{D}(j,k) = \mathcal{D}'(j,k)$. Let us apply this method to the graph in Figure 1b. From graph theory, it is known that the vertices of an acyclic graph are strictly partially ordered. There exist algorithms for constructing some linear order for a strictly partially ordered graph. Informally, a linear order is when all the vertices of a graph are arranged in a line, and all edges are directed from left to right. Any such algorithm is suitable because:
- the first vertex is always $(1,1)$ of the original graph,,
- the last vertex is always (n,m),
- when evaluating the recursive function according to a linear order, its arguments have already been evaluated.

Let us choose a row-by-row traversal. In this case, for a pair (j,k), all pairs (j',k') with $j' < j$ or $k' < k$ have already been considered. We will traverse the elements of the matrix of interval variables T in the rows from left to right and from the top row to the bottom one.

1. Let us prove the statement P_1. From the system of inequalities (2)-(5), it follows that

$$T(1,1) = [p_{1,1}, p_{1,1}] \cap T^0(1,1).$$

From the definition of the recursive function, it follows that

$$T'(1,1) = \mathcal{T}(1,1) = [p_{1,1}, p_{1,1}] \cap T^0(1,1).$$

Therefore, we have $T(1,1) = T'(1,1)$.
2. Let us assume that the statements $P_1, \ldots, P_{i-1}$ are true. We now prove that he statement P_i is also true. Among the 4 groups of relationship types described above, the first type is considered for $j = 1$, $k = 1$.
3. Let us analyze the remaining 3 types.

3a) Processing of the 1st job on the kth machine ($j = 1$, $1 < k \leq m$):
In this case, the problem satisfies the equality

$$T(1,k) = (T(1,k-1) + [p_{1,k}, p_{1,k}]) \cap T^0(1,k).$$

In turn,

$$\mathcal{T}(1,k) = (\mathcal{T}(1,k-1) + [p_{1,k}, p_{1,k}]) \cap T^0(1,k)$$

and due to the condition $\mathcal{T}(1,k-1) = T(1,k-1)$, we get

$$\mathcal{T}(1,k) = T(1,k-1) + [p_{1,k}, p_{1,k}] = T(1,k).$$

Therefore, we have $T(1,k) = T'(1,k)$.

3b) Processing of job j on the 1st machine ($1 < j \leq n$, $k = 1$): In this case, the problem satisfies the equality

$$T(j,1) = (T(j-1,1) + [p_{j,1}, p_{j,1}]) \cap T^0(j,1).$$

In case of recursion,

$$\mathcal{T}(j,1) = (\mathcal{T}(j-1,1) + [p_{j,1}, p_{j,1}]) \cap T^0(j,1).$$

 doi:10.20944/preprints202411.2070.v1

Due to the condition $\mathcal{T}(j-1,1) = T(j-1,1)$, finally we get

$$\mathcal{T}(j-1,1) = T(j-1,1) = T(j,1).$$

Therefore, we have $T(j,1) = T'(j,1)$.

3c) Processing of job j on the kth machine $(1 < j \leq n,\ 1 < k \leq m)$ assumes that the problem satisfies the equalities

$$\mathcal{T}(j-1,k) = T(j-1,k),\ \mathcal{T}(j,k-1) = T(j,k-1).$$

In this case,

$$\begin{aligned} \mathcal{T}(j,k) &= (\max\{\mathcal{T}(j-1,k), \mathcal{T}(j,k-1)\} + [p_{j,k}, p_{j,k}]) \cap T^0(j,k) \\ &= (\max\{T(j-1,k), T(j,k-1)\} + [p_{j,k}, p_{j,k}]) \cap T^0(j,k) = T(j,k). \end{aligned}$$

Therefore, we have $T(j,k) = T'(j,k)$.

All possible situations have been analyzed.

□

The main optimization problem in the PFSP is to find the order of job processing so that for some selected criterion, the function value is optimal. The most common criterion is the makespan (i.e., the total time it takes to complete the whole set of jobs). We remind that the vast majority of flow shop problems are treated as a PFSP, see for example [9–11].

To effectively solve the problem of calculating an optimal schedule, it is necessary to switch from interval recursive functions (8) to scalar recursive functions $C : \mathcal{J} \times \mathcal{M} \to N$. Consider the case when all elements of the matrix of initial time intervals are equal to $[0, T_{max}]$. In this case, all the cases of interval intersections shown in Figure 3 will be reduced to case d and will look like in Figure 4. The value of $\overline{T}(j,k)$ will always be equal to T_{max} and the interval value can be replaced by a scalar equal to the lower value of the interval $\underline{T}(j,k)$. Then the recursive function (8) will look as follows:

$$C(j,k) = \begin{cases} p_{1,1}, & j = 1,\ k = 1; \\ \underline{\mathcal{T}}(1,k-1) + p_{1,k}, & j = 1,\ k > 1; \\ \underline{\mathcal{T}}(j-1,1) + p_{j,1}, & j > 1,\ k = 1; \\ \max(\underline{\mathcal{T}}(j-1,k), \underline{\mathcal{T}}(j,k-1)) + p_{j,k}, & j > 1,\ k > 1. \end{cases} \tag{9}$$

or finally

$$C(j,k) = \begin{cases} p_{1,1}, & j = 1,\ k = 1; & a) \\ C(1,k-1) + p_{1,k}, & j = 1,\ 1 < k \leq m; & b) \\ C(j-1,1) + p_{j,1}, & 1 < j \leq n,\ k = 1; & c) \\ \max\{C(j-1,k), C(j,k-1)\} + p_{j,k}, & 1 < j \leq n,\ 1 < k \leq m. & d) \end{cases} \tag{10}$$

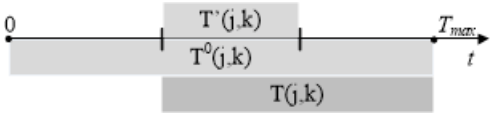

Figure 4. The case of interval intersection when $T^0(j,k) = [0, T_{max}]$

The function $C(j,k)$ calculates the completion time of job j on the kth machine. The scalar function (10) is obtained from the interval function (8) by replacing the interval with the lowest value of the interval.

 doi:10.20944/preprints202411.2070.v1

In Example 2 above, $C(j,k)$ computes only one schedule with minimum completion time, which corresponds to the "Schedule 1" diagram in Figure 2.

The analysis of formula (10) shows that when calculating the function $C(n,m)$, the traversal of pairs (j,k) is carried out in the order indicated in Figure 5a.

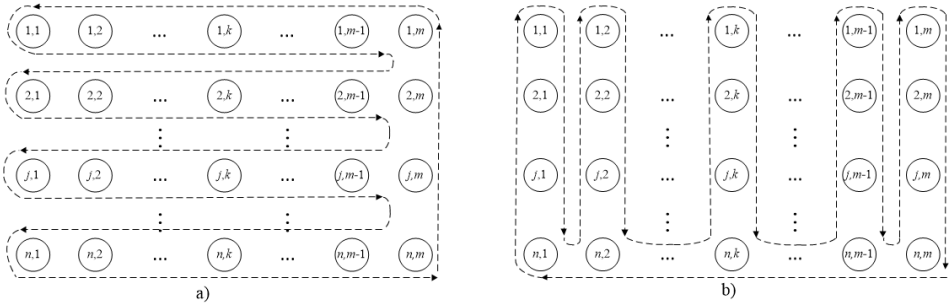

Figure 5. Two traversal options defined by two types of a recursive function

However, if the function (10) is replaced by an equivalent function (11) (the difference is in the expression 11*d*), then the pairs will be traversed according to the scheme in Figure 5b.

$$C(j,k)=\begin{cases} p_{1,1}, & j=1,\ k=1; \quad a)\\ C(1,k-1)+p_{1,k}, & j=1,\ 1<k\le m; \quad b)\\ C(j-1,1)+p_{j,1}, & 1<j\le n,\ k=1; \quad c)\\ \max\{C(j,k-1),C(j-1,k)\}+p_{j,k}, & 1<j\le n,\ 1<k\le m. \quad d) \end{cases} \tag{11}$$

Theorem 2. *Let the PFSP include m machines and n jobs. Consider some schedule and then renumber the jobs in the order in which they are located in the schedule, i.e. consider the schedule $(1,2,...,n)$ with the new numbering. In this case, the following equality is true for any j $(1\le j\le n)$ and any k $(1\le k\le m)$:*

$$\underline{T}(j,k)=C(j,k). \tag{12}$$

The point of this theorem is that the scalar function calculates the minimum completion time for any given sequence of jobs. This follows from the principle of the function construction.

Proof. If in the formula (9) $C(j,k)$ is replaced by $\underline{T}(j,k)$, then a formula for calculating $\underline{T}(j,k)$ is obtained. The direct comparison of the resulting formula with the formula (10) for calculating $C(j,k)$ proves the validity of the theorem. □

The following important theorem follows from Theorem 2.

Theorem 3. *Let the PFSP be defined for m machines and n jobs. In addition, let Π be the set of all permutations of n jobs and $\pi^*=(\alpha_1,\alpha_2,...,\alpha_n)\in\Pi$ be the permutation corresponding to the minimum completion time of all jobs, computed as $\underline{T}(\alpha_n,m)$ for π^*. For the PFSP, such a permutation satisfies the equality*

$$\underline{T}(\alpha_n,m)=C(\alpha_n,m). \tag{13}$$

The meaning of the theorem is that the minimum completion time of all jobs, computed using interval calculations, is equal to the value of the scalar recursive function for the same permutation.

Proof. The set of all permutations Π is finite and has the cardinality $n!$. In this case, as it has been proven by Theorem 2, for any permutation $\pi\in\Pi$, the equality $\underline{T}(\alpha_n,m)=C(\alpha_n,m)$ holds. Therefore, it also holds for the permutation π^*. □

Let us look at Example 2 again. If, after calculating the intervals in Table 2, we introduce a constraint of the form $r_3=7$ (the moment the job is received for processing), this will lead to a

 doi:10.20944/preprints202411.2070.v1

change in the intervals for the 3rd job as follows: $[10,10],[12,12],[14,14]$, and for the 4th job to $[13,13],[14,14],[16,16]$. Among the schedules given in the example in Figure 2, only "Schedule 3" will remain feasible. The other schedules will become infeasible. Evaluating the recursive function will not result in a feasible schedule. This example demonstrates the importance of including constraints in the initial value matrix when evaluating the recursive function.

Theorem 3 states that when computing an optimal schedule for the PFSP, one can move from interval calculations to the scalar recursive function $C(j,k)$ (10), although the cardinality of the set of values of the function $C(j,k)$ is less than or equal to the cardinality of the set of values of the function $\mathcal{T}(j,k)$. If the function $\underline{\mathcal{T}}(n,m)$ has several minima, then the function $C(n,m)$ will have the same number of minima.

Denote by the permutation $\pi=(\alpha_1,\alpha_2,\ldots,\alpha_n)$ the order of job numbers, and α_i, $i=1,2,\ldots,n$, is the number of the job at position α in the schedule (permutation) π. The recursive function was defined before to have a fixed order of the processing of the jobs. Let us add a permutation π to the definition of the function as a global variable so that we can change the order of jobs. We define new functions: $\hat{\mathcal{T}}:\hat{\mathcal{J}}\times\mathcal{M}\to\mathcal{I}$ and $\hat{C}:\hat{\mathcal{J}}\times\mathcal{M}\to N$. Here $\hat{\mathcal{J}}$ is the set of job position numbers and $\alpha\in\hat{\mathcal{J}}$. We include the permutation π in the definitions of the recursive functions (8) and (10) so that we can change the order of processing the jobs:

$$\hat{\mathcal{T}}(\alpha,k)=\begin{cases}[p_{\pi(1),1},T_{max}], & \alpha=1,\ k=1;\\ \hat{\mathcal{T}}(1,k-1)+[p_{\pi(1),k},p_{\pi(1),k}], & \alpha=1,\ 1<k\le m;\\ \hat{\mathcal{T}}(\alpha-1,1)+[p_{\pi(\alpha,1},p_{\alpha,1}], & 1<\alpha\le n,\ k=1;\\ \max\{\hat{\mathcal{T}}(\alpha-1,k),\hat{\mathcal{T}}(\alpha,k-1)\}+[p_{\pi(\alpha),k},p_{\pi(\alpha),k}], & 1<\alpha\le n,\ 1<k\le m.\end{cases}\tag{14}$$

$$\hat{C}(\alpha,k)=\begin{cases}p_{\pi(1),1}, & \alpha=1,\ k=1;\\ \hat{C}(1,k-1)+p_{\pi(1),k}, & \alpha=1,\ 1<k\le m;\\ \hat{C}(\alpha-1,1)+p_{\pi(\alpha),1}, & 1<\alpha\le n,\ k=1;\\ \max\{\hat{C}(\alpha-1,k),\hat{C}(\alpha,k-1)\}+p_{\pi(\alpha),k}, & 1<\alpha\le n,\ 1<k\le m.\end{cases}\tag{15}$$

In the function $\hat{C}(\alpha,k)$, where α, $\alpha=1,2,\ldots,n$, is the number of the position in the schedule (permutation) π and k, $k\in\mathcal{M}$, is the number of the machine, are the arguments of the recursive function, and the processing times $p_{j,k}$ and the permutation π are the parameters of the function.

Example 3. *Here we calculate the completion times of the jobs 1 and 2 on the machines 1 and 2. It should be noted that in this case the first argument of the function $\hat{C}$ is the number of the position in the shedule π.*
$\hat{C}(1,1)=p_{\pi(1),1}$;
$\hat{C}(1,2)=\hat{C}(1,1)+p_{\pi(1),2}=p_{\pi(1),1}+p_{\pi(1),2}$;
$\hat{C}(2,1)=\hat{C}(1,1)+p_{\pi(2),1}=p_{\pi(1),1}+p_{\pi(2),1}$;
$\hat{C}(2,2)=\max\{\hat{C}(1,2),\hat{C}(2,1)\}+p_{\pi(2),2}$
$\quad=\max\{p_{\pi(1),1}+p_{\pi(1),2},p_{\pi(1),1}+p_{\pi(2),1}\}+p_{\pi(2),2}$.

For simplicity of the description, we have discussed the approach for the PFSP. It can be noted that one can extend this procedure to the FSP with possibly different job sequences on the machines:

4. Implementation of the Branch and Bound Method for the PFSP

It should be noted that the PFSP was one of the first combinatorial optimization problems for which the branch and bound (B&B) method was applied [6] soon after its development [12]. A large number of works were devoted to the application of the B&B method for solving the PFSP. The papers [10,13] review the most important works on B&B algorithms for the PFSP. Since the branch and bound method follows very organically from the recursive representation of the PFSP, we consider it subsequently.

 doi:10.20944/preprints202411.2070.v1

Evaluation of Lower Bound LB1. Taking into account the permutation π of jobs, for a certain pair (α, k), the lower bound LB can be determined as follows:

$$LB(\alpha,k) = \hat{C}(\alpha,k) + \sum_{i=k+1}^{m} p_{\pi(\alpha),i} + \sum_{i=\alpha+1}^{n} p_{\pi(i),m}. \tag{16}$$

Remark 1. *Calculating the value of the function $\hat{C}(\alpha,k)$ from the value of the function $\hat{C}(\alpha-1,k)$ is simpler than calculating $LB(\alpha,k)$. Therefore, it makes sense to calculate the lower bound only for the pair (α,m). In [14], the effectiveness of the comparison with the lower bound on the last machine was also justified.*

In this case, it is possible to move from formula (16) to the formula for calculating the bound on the last machine

$$LB(\alpha,m) = \hat{C}(\alpha,m) + \sum_{i=\alpha+1}^{n} p_{\pi(i),m}. \tag{17}$$

Next, let us formulate the exact meaning of the bound $LB(\alpha,k)$. In the described case, it means that with a fixed order of a part of the jobs $1,2,\ldots,\alpha$, the processing times of all jobs cannot be less than $LB(\alpha,k)$ for any arrival order of the remaining jobs $\alpha+1,\alpha+2,\ldots,n$.

Let Π be the set of permutations of n jobs for which the value $\hat{C}(\alpha_n,m)$ has already been calculated. Then the lower bound $LB1$ for the considered job sequences will be

$$LB1 = \min_{\pi\in\Pi} \hat{C}(\alpha_n,m).$$

Thus, if the inequality

$$LB(\alpha,k) \geq LB1,$$

is satisfied for the current pair (α,k), then this sequence of jobs $1,2,\ldots,\alpha-1,\alpha$ is excluded from consideration as not promising, and the corresponding branch of permutations is cut off. The next one will be one of the sequences $1,2,\ldots,\alpha-1,\pi'(1)$ with

$$\pi' \in \mathcal{P}(\alpha,\alpha+1,\ldots,n) \; and \; \pi'(1) \neq \pi(\alpha),$$

where $\mathcal{P}(\alpha,\alpha+1,\ldots,n)$ is the set of all permutations of the $n-\alpha+1$ remaining jobs.

Branching. First of all, the branching must be consistent with the traversal defined by the recursive function. If

$$LB(\alpha,k) \geq LB1,$$

it is necessary to change the order of the jobs. However, it is necessary to change the order in such a way as to minimize the loss of information when calculating the functions $\hat{C}(1,1),\hat{C}(1,2),\ldots,\hat{C}(\alpha,m)$. The literature describes many permutation algorithms with different characteristics. Let $\mathcal{P}(1,\ldots,s) = \{\pi_1,\pi_2,\ldots,\pi_{s!}\}$. $x\mathcal{P}(\pi)$ assigns to each permutation of the set an element x, i.e., $x\mathcal{P}(\pi) = \{x\pi_1,x\pi_2,\ldots,x\pi_{s!}\}$. In this case, the algorithm for generating permutations should calculate them in such a sequence that they satisfy the following recursive property:

$$\mathcal{P}(j_1,j_2,\ldots,j_s) = \bigcup_{i=1}^{s} j_i\mathcal{P}(j_1,j_2,\ldots,j_{i-1},j_{i+1},\ldots,j_s). \tag{18}$$

This property, after calculating the function $C(j,k)$, allows one to study all permutations of the "tail" of the queue of jobs being scheduled $\mathcal{P}(j+1,j+2,\ldots,n)$ and only then track back.

Example 4. *Figure 6 shows an example of a complete permutation tree that satisfies Property (18) for $n=4$. The root of the tree is an auxiliary vertex, and the permutation corresponds to some path from the root. The*

 doi:10.20944/preprints202411.2070.v1

vertical edge shows the next level of recursion. It connects the initial sequence to its tail. The tree is traversed from top to bottom and from left to right. In this example, the permutations will be generated in the following order: (1,2,3,4),(1,2,4,3),(1,3,2,4),(1,3,4,2),...,(4,3,1,2),(4,3,2,1). There are 24 permutations in total.

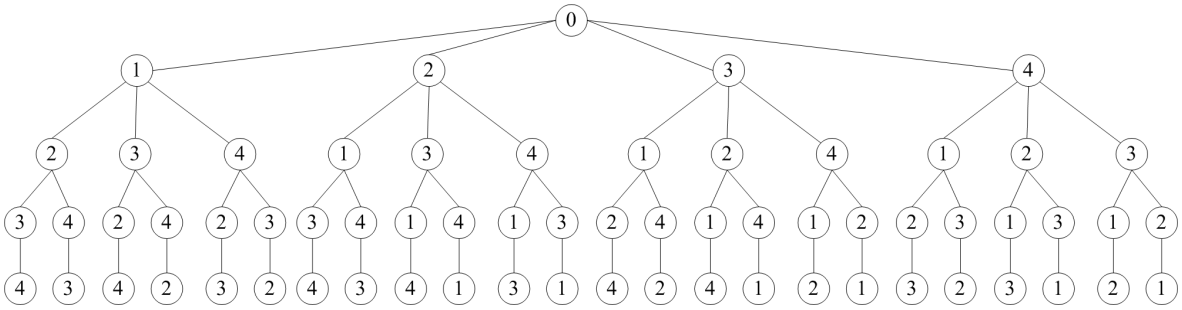

Figure 6. A complete permutation tree satisfying Property (18)

This algorithm can be implemented through recursive copying of the tail of permutations. This is a costly operation, but the algorithm can be modified so that the permutation π exists in a single instance as a global variable. This is done using forward and reverse permutations of two jobs (when returning along the search tree). The description of the B&B algorithm is given in Appendix A.

5. "And" Function

The flow shop problem discussed above is more a subject of theoretical research and has limited applicatiosn in practice. In order to bring it closer to practice, various additional elements can be introduced, first of all, various types of constraints. In real production, the main machine chain is served by various preparatory operations, for example: those associated with delivery from the warehouse to the production line; preparatory steps; technological issues; testing, etc.

Next, we expand the PFSP by introducing the *and* function. A well-known construction, shown in Figure 7a, denotes a precedence relation in which the execution of a job on machine 3 can only begin after the completion of the jobs on machines 1 and 2. Using the *and* function, it will look like in Figure 7b. In what follows, we will interpret the *and* function as a machine with zero processing time for any job. Thus, the chained precedence graph for the PFSP is expanded into a tree precedence graph when using the *and* function. Thus, the PFSP is transformed towards an Assembly Line job (AL), but without the presence of a conveyor belt and workstations. This problem can be called the Assembly Permutation Flow Shop problem (APFSP) . The absence of a conveyor belt frees from the "cycle time" limitation, and the absence of workstations frees from solving the balancing problem, i.e., an optimal distribution of the jobs among the stations, taking into account the cycle time.

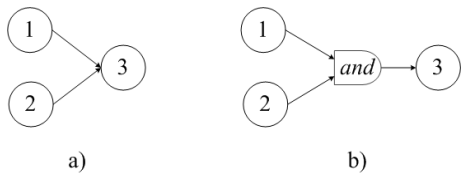

Figure 7. Function *and*

Despite the absence of assembly line attributes, this model can have a wide application.
The function *and* in the interval representation $\mathcal{T}_{and} : \mathcal{J} \times \mathcal{M} \to \mathcal{I}$ looks as follows:

$$\mathcal{T}_{and}(j,k) = \max\{\mathcal{T}(j, pred_1(k)), \mathcal{T}(j, pred_2(k))\},$$

or

$$\mathcal{T}_{and}(j,k) = [\max\{\underline{\mathcal{T}}(j, pred_1(k)), \underline{\mathcal{T}}(j, pred_2(k))\}, \max\{\overline{\mathcal{T}}(j, pred_1(k)), \overline{\mathcal{T}}(j, pred_2(k))\}],$$

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 27 November 2024 doi:10.20944/preprints202411.2070.v1

or in a scalar expression $C_{and} : \mathcal{J} \times \mathcal{M} \rightarrow N$.

$$C_{and}(j,k) = \max\{C(j, pred_1(k)), C(j, pred_2(k))\}. \tag{19}$$

Accordingly, we get the function $\hat{C}_{and} : \hat{\mathcal{J}} \times \mathcal{M} \rightarrow N$, where $\hat{\mathcal{J}}$ is a set of job position numbers and $\alpha \in \hat{\mathcal{J}}$:

$$\hat{C}_{and}(\alpha,k) = \max\{\hat{C}(\alpha, pred_1(k)), \hat{C}(\alpha, pred_2(k))\}. \tag{20}$$

Including the *and* function into the PFSP causes the machines to process the job according to the precedence tree. To set a specific order of the precedence graph traversal, we assume that the function $pred_1$ is always the first one in the formula (19) and determines the upper arc in Figure 7b. Adding the *and* function causes the assumption in Section 3 to be invalid: "a job cannot be processed by more than one machine at a time", since the machines 1 and 2 in Figure 7b can execute the same job simultaneously.

It can be proven that when adding the function *and* to the PFSP (i.e., $C(j,k) = C_{and}(j,k)$ and $\hat{C}(\alpha,k) = \hat{C}_{and}(\alpha,k)$), Theorem 3 remains valid, and the function $C(j,k)$ supplemented by $C_{and}(j,k)$ also implements a greedy algorithm for a fixed sequence of jobs. Figure 8a shows the example of a precedence graph between the machines. Figure 8b shows the precedence graph between machines and jobs (with a fixed order of jobs). Figure 8c shows the computation (superposition) graph of the recursive function for a certain sequence of 4 jobs. In the latter case, the arc determines the transfer of the value $C(j,k)$.

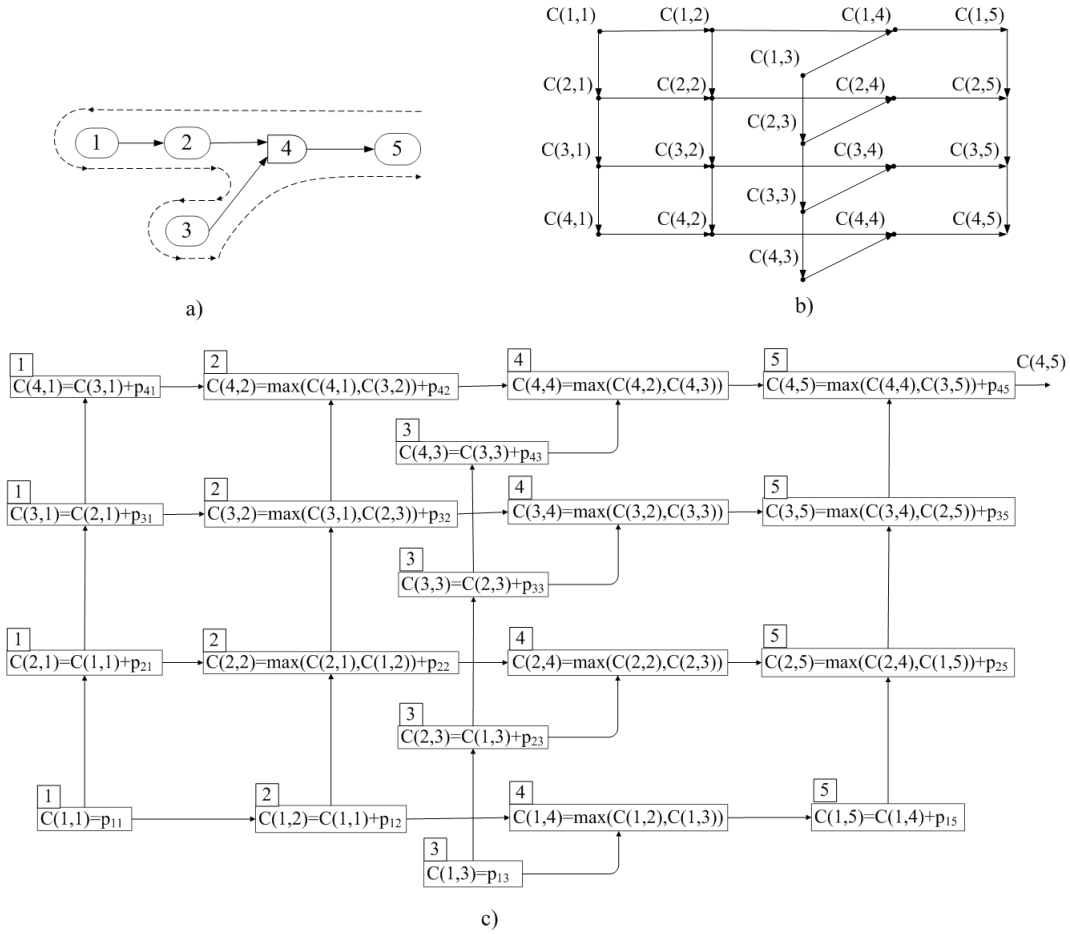

Figure 8. An example of a PFSP graph with the *and* function

Adding the *and* function to the PFSP causes the machines to process a job according to the precedence tree. Figure 8 shows an example of a precedence graph between the machines (Figure 8a),

 doi:10.20944/preprints202411.2070.v1

the corresponding precedence graph between the machines and jobs (Figure 8b), and the graph of computation (superposition) of the recursive function for some sequence of 4 jobs. The graph in Figure 8b is acyclic. In this case, it is convenient to traverse the graph according to the jobs, i.e., the 1st job tree, the 2nd job tree, etc. in accordance with the job permutation algorithm. The tree of each job is traversed as indicated in the figure by the dotted line.

6. Implementation of the Branch and Bound Method for the APFSP

When implementing any method for the APFSP, it is necessary to switch to a precedence graph representation with typed vertices. Let us define the set $\{sop, op, and\}$ as the following types of vertices:

sop – the starting vertex of the graph (machine);
op – intermediate or final vertex of a graph (machine);
and – the vertex corresponding to the and function.

Let us define the function $Type : \mathcal{M} \to \{sop, op, and\}$ that gives the type of a vertex. It is necessary to define explicitly the precedence functions for the vertices $pred$, $pred_1$, $pred_2 : \mathcal{M} \to \mathcal{M}$.

$pred$ is defined for the vertices of type sop, $k_1 = pred(k_2)$ means that the vertex k_1 precedes the vertex k_2.
$pred_1$ and $pred_2$ are defined for the vertices of type and, $k_1 = pred_1(k_3)$ means that the vertex k_1 precedes the vertex k_3 along the first incoming edge, $k_2 = pred_2(k_3)$ means that the vertex k_2 precedes the vertex k_3 along the second incoming edge.

The main issues of the implementation of the method are also the choice of the lower bounds (LB) calculating techniques and branching algorithms. Consider the calculation of the lower bound LB1 by traversing the jobs, that is, similar to Figure 5, but with tree traversal. This means that formulas (10) and (19) are used.

Evaluation of the Lower Bound LB1. The analysis of the formula (10) shows that when calculating the function $C(n, m)$, the operations (j, k) are traversed in the order indicated in Figure 5a. Consider the example in Figure 8a. If the and function is present, the job processing tree will be traversed as shown in the figure. Naturally, the value of $LB(\alpha, k)$ is calculated after calculating $\hat{C}(\alpha, k)$. The values of LB for the job α $(1 < \alpha < n)$ considering that $p_{\alpha,4} = 0$ can be calculated as follows:
$LB(\alpha, 1) = \hat{C}(\alpha, 1) + p_{\alpha,2} + p_{\alpha,4} + p_{\alpha,5} + p_{\alpha+1,5} + ... + p_{4,5}$;
$LB(\alpha, 2) = \hat{C}(\alpha, 2) + p_{\alpha,4} + p_{\alpha,5} + p_{\alpha+1,5} + ... + p_{4,5}$;
$LB(\alpha, 3) = \hat{C}(\alpha, 3) + p_{\alpha,4} + p_{\alpha,5} + p_{\alpha+1,5} + ... + p_{4,5}$;
$LB(\alpha, 4) = \hat{C}(\alpha, 4) + p_{\alpha,5} + p_{\alpha+1,5} + ... + p_{4,5}$;
$LB(\alpha, 5) = \hat{C}(\alpha, 5) + p_{\alpha+1,5} + ... + p_{4,5}$.
From the example considered, it is clear that the calculation of $LB(\alpha, k)$ is carried out using the same tree traversal as when calculating $\hat{C}(\alpha, k)$. In this case, for $LB(\alpha, k)$, Equality (16) is transformed into the equality

$$LB(\alpha, k) = \hat{C}(\alpha, k) + \sum_{i \in U_{\alpha,m}} p_{\pi(\alpha),i} + \sum_{i=\alpha+1}^{n} p_{\pi(i),m},$$

where $U_{\alpha,m}$ is the set of vertices belonging to the path from α to m in the machine precedence graph. The uniqueness of this path follows from the fact that the machine precedence graph is a tree, and m is its root node (see, for example, Figure 8a or b). Taking into account Remark 1, the expression (17) can also be used.

The Appendix presents three algorithms used for the solution of the problem AFS.

The described algorithms work both for the PFSP and the APFSP. The only difference is in the definitions of the $pred$ function. In the case of the PFSP, $pred(k) = k - 1$. For the APFSP, the precedence functions are determined from the precedence graph or a matrix. In this case, it is obvious that for

 doi:10.20944/preprints202411.2070.v1

the APFSP, the complexity of the B&B algorithm does not increase. An important feature should be noted. In the case when the last vertex in the precedence graph is the *and* function, the B&B algorithm executes correctly, but its "predictive power" is reduced to zero.

Recursive functions allow one to calculate also the values of objective functions different from makespan. This can be done by replacing the function $\hat{C}(\alpha, k)$ by

$$C^*(\alpha, k) = \hat{C}(\alpha, k); < calculating\ the\ value\ of\ the\ objective\ function > .$$

In this case, the calculation of the optimization criterion is concentrated in the body of the function $C^*(\alpha, k)$ and is invariant to the details of the optimization method.

7. Evaluating the Effectivenes of the Algorithm

The existing publications on the B&B method in the scheduling theory use tests that are characterized by the processing times on specific machines. To assess the effectiveness of the B&B method, without using pure processing time indicators, it is advisable to determine how much less of some elementary calculations were completed due to the fact that some sets of job permutations were deliberately discarded and not checked. There are two possible approaches here:

- consider it elementary to calculate the completion time of the job's last technological operation (approach L);
- consider it elementary to calculate the completion time of each technological operation of each job (approach A).

In both cases, it is necessary to determine the maximum possible number of such calculations when solving a problem for n jobs and m machines, and then to compare it with the number of calculations actually performed.

In the L approach, the maximum number of elementary computations is determined by the permutation generation algorithm (see Appendix). At each step, this algorithm either adds a new job to the already fixed initial part of the vector π, or replaces the last job in this fixed part by the next one. In both cases, the completion time of the last technological operation of the last job in the fixed part of π is calculated. The completion times of all technological operations of all previous jobs are already known and do not need to be calculated again. Thus, an "elementary computation" is the calculation of m elements of the job completion times matrix (T_{op}) row, and the total number of such computations is equal to the number of different variants of the initial part of the vector π generated by the algorithm.

Consider, for example, the generation of permutations and the computation of the completion times for three jobs (n =3). If the rejection of branches of the search along the lower bound LB is never performed, then the sequence of actions performed by the algorithm is described by Table 3.

 doi:10.20944/preprints202411.2070.v1

Table 3. Sequence of actions of the algorithm for n=3 without the rejection of branches.

Recursion level	Algorithm argument	π	π fixed part	π "tail"	Computed T_{op} row
0	1	[1,2,3]	[]	[1,2,3]	–
1	2	[1,2,3]	[1]	[2,3]	1
2	3	[1,2,3]	[1,2]	[3]	2
3	4	[1,2,3]	[1,2,3]	[]	3
2	3	[1,3,2]	[1,3]	[2]	2
3	4	[1,3,2]	[1,3,2]	[]	3
1	2	[2,1,3]	[2]	[1,3]	1
2	3	[2,1,3]	[2,1]	[3]	2
3	4	[2,1,3]	[2,1,3]	[]	3
2	3	[2,3,1]	[2,3]	[1]	2
3	4	[2,3,1]	[2,3,1]	[]	3
1	2	[3,2,1]	[3]	[2,1]	1
2	3	[3,2,1]	[3,2]	[1]	2
3	4	[3,2,1]	[3,2,1]	[]	3
2	3	[3,1,2]	[3,1]	[2]	2
3	4	[3,1,2]	[3,1,2]	[]	3

As a result of the algorithm's operation, 15 initial parts of the vector π were generated (including 6 complete permutations of three tasks), and 15 elementary computations of the T_{op} rows were performed. With the size h of the initial part of the vector π $(1 \leq h \leq n)$, the total number of such generated initial parts is equal to the number of placements from n to h:

$$A_n^h = \frac{n!}{(n-h)!}.$$

The total number of initial parts of the vector π of all possible sizes from 1 to n is equal to

$$S_{max}(n) = \sum_{h=1}^{n} A_n^h = \sum_{h=1}^{n} \frac{n!}{(n-h)!}.$$

When using the B&B method, the algorithm will perform all S_{max} elementary calculations extremely rarely. However, this estimate is achievable in practice. Let us consider the degeneration case in which all rows of the matrix $p_{j,*}$ are identical. In this case, the LB value calculated using formulas (16) or (17) will be the same for any values of the arguments, and none of the branches will be discarded. The number of elementary calculations performed by the algorithm will be exactly S_{max}. In approach A, the maximum possible number of elementary calculations is m times $S_{max}(n)$ since every row of the job completion times matrix contains m elements. In the optimized algorithm, the number of real calculations will be less than the given values. The effectiveness of solving a problem can be assessed by the proportion of calculations not performed. If $S_L(i)$ and $S_A(i)$ are the numbers of calculations when solving problem number i for n jobs and m machines using the approaches L and A, respectively, then the efficiency of the solutions can be estimated using the following formulas:

$$E_L(n,i) = \frac{S_{max}(n) - S_L(i)}{S_{max}(n)},$$

$$E_A(n,m,i) = \frac{mS_{max}(n) - S_A(i)}{mS_{max}(n)}.$$

A zero value of these indicators means that the use of the method did not provide any gain in the number of calculations; a theoretically unattainable value of 1 (or 100%) means that the use of the method completely eliminated the need for calculations.

 doi:10.20944/preprints202411.2070.v1

The authors performed 10,000 tests for problems with 5–11 jobs on random acyclic graphs with random job processing times. The problem generation parameters in these experiments were as follows:

- total number of nodes in the acyclic graph (technological operations): 15;
- number of initial operations: random value in [1,3];
- number of ”and” operations: random value in [1,4];
- job processing times ($p_{j,k}$): random value in [1,10].

All random values are uniformly distributed integers. For each number of jobs, a histogram of the number of experiments distribution by the ranges of the E_L efficiency obtained in the experiment was constructed. Thus, 7 histograms were constructed, presented in Figure 9.

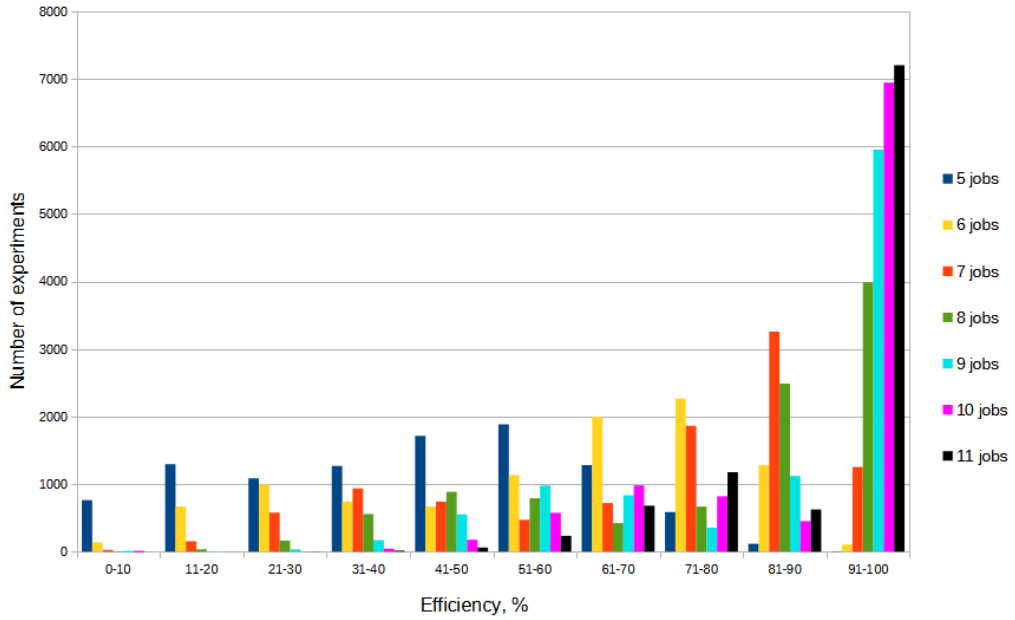

Figure 9. Plots of the numerical experiments

The histograms in the figure show how many times a particular E_L efficiency was achieved during the random experiments. The horizontal axis shows the ten-percent efficiency ranges (from ”0–10” to ”91–100”), and the vertical axis shows the number of experiments out of 10,000 performed in which the corresponding efficiency value was achieved. It can be seen from the figure that as the number of jobs increases, the efficiency of the algorithm also increases. The histogram for five jobs has a maximum around 50%. This means that most of the experiments with five jobs showed a fifty percent efficiency, that is, approximately half of the maximum possible number of elementary calculations for a given task were performed. However, a significant number (about 800) of experiments showed an efficiency of less than 10%. As the number of jobs increases, the histogram maximum moves into the high efficiency region. For eleven jobs, more than 7,000 out of 10,000 experiments performed showed an efficiency in the range of 91–100%, i.e., less than 10% of the number of elementary calculations that would be required to solve the problem when analyzing all possible permutations were performed.

8. Conclusion

This article used the simplest form of a lower bound. A detailed overview of lower bounds for the B&B method for the PFSP has been given in [9]. The main result of this study seems to be an organic transition from the PFSP to the more general and relevant APFSP using recursive functions.

The article dealt with tree precedence graphs, but it is obvious that the described method is applicable also to other acyclic graphs.

 doi:10.20944/preprints202411.2070.v1

The article did not address the issue of computational efficiency. Recursive functions are not efficient in a direct computation. However, there exist calculation transformations, for example: using previously calculated values (calculation with memorization), switching from recursion to iteration, avoiding the use of a memory stack, etc., which can significantly increase the efficiency.

Further research will be focused on the development of:

- heuristic and metaheuristic algorithms that can be embedded in the given scheme, for example, genetic algorithms or the NEH heuristic, etc.;
- new recursive functions that will bring the model closer to reality; there is such a prospect, but its presentation does not fit into the framework of this article.

Appendix

This appendix contains the algorithms for finding an optimal permutation.

Algorithm 1 for finding an optimal permutation (with definitions of the variables)

Begin Algorithm 1. (Initializing and calling PermutationB&B).

1. π is a vector of job permutations, a global variable.
2. P is the matrix of job processing times on each machine.
3. $T_{op} \leftarrow 0$ ' matrix of the job completion times.
4. $\pi \leftarrow (1, 2, ..., n)$ ' the initial order of the jobs.
5. $LB1 \leftarrow$ the maximum value.
6. $PermutationB\&B(1)$ ' calling a permutation set generator.

End Algorithm 1.

Algorithm 2 for calculating the function $\hat{C}(\alpha, k)$ for three types of graph vertices.

Begin Algorithm 2. (Calculation$\hat{C}(\alpha, k)$).

1. $if\ Type(k) ='' op''\ then$
2. $\quad if\ \alpha > 0\ then$ 'Not the first job
3. $\quad\quad C \leftarrow \max\{\hat{C}(\alpha - 1, k), \hat{C}(\alpha, pred(k))\} + P_{\pi(\alpha),k}$
4. $\quad else$ 'The first job
5. $\quad\quad C \leftarrow \hat{C}(\alpha, pred(k)) + P_{\pi(\alpha),k}$
6. $\quad end\,if$
7. $else$
8. $\quad if\ Type(k) ='' sop''\ then$
9. $\quad\quad if\ \alpha > 0\ then$
10. $\quad\quad\quad C \leftarrow \hat{C}(\alpha - 1, k) + P_{\pi(\alpha),k}$
11. $\quad\quad else$
12. $\quad\quad\quad C \leftarrow P_{\pi(\alpha),k}$
13. $\quad\quad end\,if$
14. $\quad else$
15. $\quad\quad if\ Type(k) ='' and''\ then$
16. $\quad\quad\quad C \leftarrow \max\{\hat{C}(\alpha, pred_1(k)), \hat{C}(\alpha, pred_2(k))\}$
17. $\quad\quad end\,if$
18. $\quad end\,if$
19. $end\,if$
20. $T_{op}(\alpha, k) \leftarrow C$

 doi:10.20944/preprints202411.2070.v1

End Algorithm 2.

Algorithm 3 generates a set of permutations satisfying Property (18), including the iteration of the B&B method.

Begin Algorithm 3. (Recursive algorithm B&B for roblem AFS $PermutationB\&B(\alpha)$)

1. ' α – the sequential number of the job in the vector π
2. *if* $\alpha > 1$ *then*
3. $\quad LB \leftarrow \hat{C}(\alpha - 1, m) + \sum_{i=\alpha}^{n} p_{\pi(\alpha),m}$
4. $\quad$ *if* $LB > LB1$ *then exit*
5. *end if*
6. *if* $\alpha = n + 1$ *then*
7. $\quad LB1 \leftarrow LB$
8. $\quad$ *exit*
9. *end if*
10. $PermutationB\&B(\alpha + 1)$
11. *for* $i = \alpha + 1$ *to* n
12. $\quad \pi(\alpha) \leftrightarrow \pi(i)$ 'swap the elements α and i of the vectorπ
13. $\quad PermutationB\&B(\alpha + 1)$
14. $\quad \pi(\alpha) \leftrightarrow \pi(i)$ 'return the elements α and i of the vector to their previous positions π
15. *next* i

End Algorithm 3.

References

1. Lai, T. A note on the heuristics flowshop scheduling. *Journal of the Operational Research Society* **1994**, *44*, 648–652.
2. Henneberg, M.; Neufeld, J. A constructive algorithm and a simulated annealing approach for solving flow shopproblems with missing operations. *International Journal of Production Research* **2016**, *54-12*, 3534–3550.
3. Ruiz, R.; Maroto, C.; Alcaraz, J. Two new robust genetic algorithms for the flowshop scheduling problem. *Omega (The International Journal of Management Science)* **2006**, *34-5*, 461–476.
4. Bargaoui, H.; Driss, O. Multi-agent model based on tabu search for the permutation flow shop scheduling problem. *istributed Computing and Artificial Intelligence* **2014**, *3-8*, 27.
5. Alekseeva, E.; Mezmaz, M.; Tuyttens, D.; Melab, N. Parallel multi-core hyper-heuristic grasp to solve permutation flow shop problem. *Concurrency and Computation: Practice and Experience* **2017**, *29-9*, e3835.
6. Ignall, E.; Schrage, L.E. Application of the branch and bound technique to some flow shop problems. *Journal of the Operational Research Society* **1965**, *13*, 400-412.
7. Kalmykov, S.A.; Shokin, U.I.; Uldashev, Z.X. *Interval analysis methods*; Science: Novosibirsk, USSR, 1986 (in Russian).
8. Lazarev, A.A.; Gafarov, E.R. *Scheduling theory. Problems and algorithms*; MSU: Moscow, Russian Federation, 2011 (in Russian).
9. Ladharia, T.; Haouaria, M. A computational study of the permutation flow shop problem based on a tight lower bound. *Computers & Operations Research* **2005**, *32*, 1831–1847.
10. Gmys, J.; Mezmaz, M.; Melab, N.; Tuyttens, D. A computationally efficient Branch-and-Bound algorithm for the permutation flow-shop scheduling problem. *European Journal of Operational Research* **2020**, *284(3)*, 814–833.
11. Belabid, J.; Aqil, S.; Allali, K. Solving permutation flow shop scheduling problem with sequence-independent setup time. *Journal of Applied Mathematics* **2020**, 1–11.
12. Land, A.H.; Doig, A.G. An automatic method for solving discrete programming problems. *Econometrica* **1960**, *28*, 427–520.
13. Potts, C.N.; Strusevich, V.A. Fifty years of scheduling: a survey of milestones. *Journal of the Operational Research Society* **2009**, *60*, 41–68.

 doi:10.20944/preprints202411.2070.v1

14. Brooks, G. H.; White, C.R. An algorithm for finding optimal or near optimal solutions to the production scheduling problem. *Journal of Industrial Engineering* **1965**, *16.* no. 1, 34-40.