S©MR ISSN 1813-3304

СИБИРСКИЕ ЭЛЕКТРОННЫЕ
МАТЕМАТИЧЕСКИЕ ИЗВЕСТИЯ

Siberian Electronic Mathematical Reports
http://semr.math.nsc.ru

*Том 16, стр. 42–84 (2019)* УДК 519.854.2
DOI 10.33048/semi.2019.16.003 MSC 90B35

# AN ALGORITHM WITH PARAMETERIZED COMPLEXITY OF CONSTRUCTING THE OPTIMAL SCHEDULE FOR THE ROUTING OPEN SHOP PROBLEM WITH UNIT EXECUTION TIMES

R.A. VAN BEVERN, A.V. PYATKIN, S.V. SEVASTYANOV

Abstract. For the Routing Open Shop problem with unit execution times, the first algorithm with parameterized complexity is designed for constructing an optimal schedule. Its running time is bounded by a function $(Pol(|V|) + f(m, g)) \cdot |I|$, where $Pol(|V|)$ is a polynomial of the number of network nodes, $f(m, g)$ is a function of the number of machines and the number of job locations, and $|I|$ is the input length in its compact encoding.

**Keywords:** $FPT$-algorithm, Open Shop problem, routing, scheduling, UET, parameterized complexity.

## 1. Introduction

The target problem of this paper is the *Routing Open Shop* problem (*ROS-problem*, for short) being an extension of two classical discrete optimization problems: the *Open Shop* problem and *metrical Travelling Salesman Problem* (*metrical TSP*).

In the **Open Shop** problem [9], one needs to perform a given set of jobs $\mathcal{J} \doteq \{J_1, \ldots, J_n\}$ on a given set of machines $\mathcal{M} \doteq \{M_1, \ldots, M_m\}$, while the execution time $p_{kj}$ of the operation of each job $J_j$ on each machine $M_k$ is known. It

van Bevern, R., Pyatkin, A.V., Sevastyanov, S.V., An algorithm with parameterized complexity of constructing the optimal schedule for the routing open shop problem with unit execution times.



The first author was supported by the Russian Foundation for Basic Research (grant 16-31-60007). The second author was supported by the Russian Foundation for Basic Research (grant 17-01-00170). The third author was supported by the Russian Foundation for Basic Research (grant 18-01-00747).

*Received October, 2, 2018, published January, 27, 2019.*

is required to construct a minimum length schedule for performing all operations, which meets the non-simultaneity requirements to pairs of related operations: **no two operations of the same machine or on the same object may overlap in time** (see Definition 2.12).

In the **metrical Travelling Salesman Problem**, we are given a complete ordinary graph $G = (V, E)$ with weighted edges, where the weight $\tau_{ij}$ of an edge $e_{ij} = (v_i, v_j)$ is a nonnegative integer representing the distance between vertices $v_i$ and $v_j$ of graph $G$. The distance function is symmetrical and meets the triangle inequality. It is required to find in $G$ a *hamiltonian cycle* of minimum weight (i.e., a route passing through all vertices, such that each vertex, but the first one, appears exactly once, and the last vertex coincides with the first one; the weight of the route is the total weight of its edges).

***The ROS-problem (Routing Open Shop)*** We are given $n$ *objects* $\{J_1, \ldots, J_n\}$ on a map, for which a certain set of *operations* should be performed by specialized executors (that will be referred to as *machines*) from a given set $\{M_1, \ldots, M_m\}$. We consider a special (and at the same time, classical) case, when **each machine** $M_k$ **performs exactly one operation** $O_j^k$ (of a given nonnegative integral duration $p_{kj}$) **on each object** $J_j$. A *schedule* for performing the operations should meet standard requirements to multistage systems. We assume that the objects are located at *nodes* of a *transportation network* $G = (V, E)$, and that the machines travel over the edges of the network. (This makes our problem setting different from the classical scheduling problems for multistage systems, where the machines are assumed to be statical — fixed in space, while the objects at which the machines should perform operations have to move between the machines either instantly, or for a given time.) All the time that a machine performs an operation on an object, it must be at the node hosting the object. The travel time of a machine through an edge of the network is **fixed** (i.e., depends neither on a machine, nor on the direction, nor on any other circumstances) and is assumed to have a nonnegative integral value. At the beginning, all machines are located at a special node called *depot*. At the completion of the whole process all operations must be performed and all machines must return to the depot.

It follows from the non-simultaneity requirement that each machine should perform its operations in a certain sequence which is to be determined in the problem solution. Besides, the order of performing the operations on an object is unfixed either and may be chosen arbitrarily. (The problems of that type are called *Open Shop* problems.) The **main goal** of the problem solution is constructing a *complete feasible schedule* $S$ which chronologically describes all machine actions (i.e., performing the operations and traveling through the network), meets all necessary requirements and provides minimum to the completion time of the whole process ($F_{\max}(S)$). According to the standard three-field notation of scheduling problems [14], we denote this problem as $\langle RO \,||\, F_{\max} \rangle$. To bring our terminology to correspondence with the standard terminology of scheduling theory for multistage systems, the set of operations on an object $J_j$ will be further called *job* $J_j$.

An alternative setting of the above problem was considered in [1, 19], where the travel times of a machine through edges of the network were interpreted as setup times of the machine dependent on the pair of consequent jobs processed by that machine (more precisely, depended on the groups to which the jobs belong) and independent of a machine.

***FPT-algorithms*** are nowadays one of the most popular and promising approaches in discrete optimization. Along with results establishing the NP-hardness of problems, development of such algorithms provides a deeper understanding of the nature of the computational complexity of hard-to-solve problems, enables one to clarify which problem parameters have the most significant impact on its complexity. The main idea behind designing such an algorithm consists of three components. First, we should detect all "critical" parameters of the problem under consideration, i.e., those ones that stipulate the exponential running time of any algorithm for the exact problem solution. (In other words, critical parameters are those ones that, while being part of the input, make the problem NP-hard.) Secondly, we should design an algorithm for the exact problem solution such that the dependence of its running time on all critical parameters could be separated from the dependence on all other parameters. And thirdly, the latter dependence should be polynomial in the input length. A problem is called *fixed-parameter tractable (FPT)* with respect to its parameter $k$, if there is an algorithm that solves any instance $I$ in $f(k)\cdot|I|^{O(1)}$ time, where $f(x)$ is an arbitrary computable function. The corresponding algorithm is normally called a *fixed-parameter algorithm*, or an *algorithm with parameterized complexity*. For more detail, we refer the reader to the recent textbook by Cygan et al. [7].

Note that a fixed-parameter algorithm running in $O(2^k\cdot|I|)$ time runs in polynomial time for $k\leq O(\log|I|)$, whereas an algorithm with running time $O(|I|^k)$ runs in polynomial time only if $k$ is constant. (At that, changing the value of that constant changes the power of the polynomial.) The latter algorithm is not a fixed-parameter algorithm.

Recently, the field of FPT-algorithmics in scheduling and routing has shown increased interest [3, 4, 10–13, 16], whereas such algorithms for problems containing elements of both routing and scheduling seem to be rare [2, 5]. The main target of our research is studying a possibility for designing an $FPT$-algorithm for the exact solution of the $ROS$-problem.

***A review of known results on the $ROS$-problem.*** If the number of machines ($m$) is part of the input, the $ROS$-problem is NP-hard in strong sense, even if the network contains a single node (because the *Open Shop* problem is strongly NP-hard [18]). On the other hand, if there is only one machine, and all operations are of zero length (while remaining obligatory), then searching for an optimal schedule is equivalent to searching for a shortest route of the machine through *active nodes* of the network (i.e., the depot and all nodes locating the objects), which is equivalent to the corresponding metrical TSP. The NP-hardness of the latter is known from [17, p.371], where it was shown that the NP-hard decision problem of existing a hamiltonian cycle in a given graph $G$ can be polynomially reduced to a metrical TSP with edge weights 1 and 2. This implies that the $ROS$-problem is strongly NP-hard, when the number of active nodes ($g$) is part of the input. Consequently, if one wishes to solve the $ROS$-problem in polynomial time, he has to restrict parameters $m$ and $g$. (In other words, the algorithm should be parameterized with respect to $m$ and $g$.)

Yet this is insufficient for obtaining a polynomial-time algorithm, since it is known [9] that even on the network with a single node in the case of only three machines the problem is NP-hard (being the three-machine Open Shop problem). Thus, we have to restrict some other parameters of the $ROS$-problem (related to

constraints of the Open Shop problem). Those parameters are the number of jobs $n$ and execution times of operations. Yet if we bound the number of jobs by a constant, the number of operations becomes also bounded by a constant. At that, the problem of founding the optimal schedule reduces to the enumeration of a constant number of variants of a mutual order of relative operations, and for each such variant — to searching for the critical path in the corresponding graph with a constant number of vertices, which yields a trivial algorithm of problem solution in polynomial time. Thus, only the problem version with restricted execution times (while the number of jobs remains part of the input) is of interest.

In the current paper, we investigate one of the most simple restrictions on the execution times of operations, assuming that all operations are of **unit length**. We call this problem *ROS-UET* (*Routing Open Shop with Unit Execution Times*).

***On the encoding schemes of the problem input.*** First we should note that the *ROS-UET* problem **is not a special case** of the *ROS*-problem, because these two problems use **different encoding schemes** of their input. (Clearly, to perform the correct complexity analysis of a discrete optimization problem, one needs to use the most compact encoding scheme of its input.) Since in the *ROS-UET* problem all objects have equal vectors of operation durations, and only differ by their locations in the transportation network, the information on the objects can be given in the input in an *aggregated form*, in which the objects located at the same node are indistinguishable. To provide this information, it is sufficient to specify *the number of the objects* located at each active node only. Thus, the encoding length of the information on the objects in the *ROS-UET* problem can be estimated from above and from below by the amounts $O(g \log n)$ and $\Omega(g + \log n)$, respectively. Such an encoding scheme for the input data of the *ROS-UET* problem will be further referred to as a *compact encoding*. The input length of an instance $I$ will be denoted as $|I|$. At that, for the *ROS-UET* problem, the compact encoding will be used.

In [2], an $FPT$-algorithm was designed for the decision version of the *ROS-UET* problem (i.e., for **computing the optimum** of the problem); the algorithm was parameterized with respect to $m$ and $g$. For the original *ROS-UET* problem (i.e., for that of **constructing an optimal schedule**) the algorithm was completed up to a $(m+g)$-parameterized algorithm with a polynomial dependence of its running time on the parameter $n$ (the number of objects), which, however (in view of the logarithmic dependence of the input length on this parameter), **did not allow** to treat it as an $FPT$-algorithm. As also noted in [16], until today it was unknown if the problem allows a polynomial-time solution when the values of parameters $m$ and $g$ are fixed, while the number of the objects ($n$) is part of the input. In the current paper, we give a positive answer to this question.

***Results of the paper.*** **The main result** of the paper is an $FPT$-algorithm for constructing an optimal schedule for any given instance of the *ROS-UET* problem. Its running time is $(Pol(|V|) + f(m, g)) \cdot |I|$, where $Pol(|V|)$ is a polynomial of the number of network nodes, $f(m, g)$ is a function of the number of machines and the number of job locations, and $|I|$ is the input length in its compact encoding. **The secondary result** is a simple $g$-parameterized linear-time $FPT$-algorithm computing **asymptotically optimal** schedules under bounded $g$, increasing $n$, and slowly increasing (as $o(n)$) parameter $m$.

**The outline of the paper by sections**:

## 2. Basic notions and notation

We notice here that this section performs a reference function. For the reader's convenience, all definitions of basic notions related to problem settings and to algorithms are systematized and collected in one place, which enables the reader to find them easily. You may skip this section at your first reading, while referring to it as necessary.

### Common mathematical notation

$\mathbb{N}$ — the set of positive integral numbers;
$\mathbb{Z}$ — the set of integers;
$\mathbb{Z}^+$ — the set of non-negative integers;
$\mathbb{R}$ — the set of real numbers;
$\{\dots\}$ — a set;
$\langle\dots\rangle$ — an ordered set (a sequence).
The same angle brackets will be used for denoting a stable *aggregate* consisting of different-type parts (e.g., a pre-schedule $P = \langle \Delta, s, T, D, RSS\rangle$); a misunderstanding cannot arise, since in a definition of a sequence, normally, an ellipsis is used (which means a variable number of components), while the number of components of an aggregate is fixed.
$[a, b] = \{t \in \mathbb{R} \mid a \le t \le b\}$ — a closed interval of real numbers;
$[a, b]_{\mathbb{Z}} = \{t \in \mathbb{Z} \mid a \le t \le b\}$ $(a, b \in \mathbb{Z},\ a \le b)$ — an interval of integers;
$[x] = [1, x]_{\mathbb{Z}} = \{1, \dots, x\}$ $(x \in \mathbb{N})$ — a starting segment of the set of integers;
$X1 \oplus X2$ — a concatenation of sequences $X1$ and $X2$;
**null** — an empty string or sequence.

**Definition 2.1** (a unit interval)**.** An interval of real numbers $u_i = [i-1, i]$, defined for an integral positive $i \in \mathbb{N}$ will be called a *unit interval*.

**Definition 2.2** (a discrete domain of time)**.** A subset of points at the time axis compound of a finite set of unit intervals is called a *discrete domain of time*. The number of different unit intervals forming a domain $\mathcal{T}$ is called a *domain length* and is denoted by $\|\mathcal{T}\|$.

**Definition 2.3** (a cyclic shift)**.** We say that a permutation $\pi'' = (\pi''(1), \ldots, \pi''(n))$ is obtained from $\pi' = (\pi'(1), \ldots, \pi'(n))$ by a *cyclic shift* by $h \in [0, n-1]_{\mathbb{Z}}$, if for any $i \in [n]$,

$$\pi''(i) = \begin{cases} \pi'(i-h), & \text{if } i > h \\ \pi'(n+i-h), & \text{otherwise.} \end{cases}$$

**Definition 2.4** (computational operations)**.** Binary operations of comparison/adding/
subtraction of two numbers, as well as unary operations of reading an address of an array element and a subsequent extraction of its value from the array will be called *elementary computational operations*. An operation of reading/writing one bit of information will be called a *bit-operation*.

**Remark 1.** *It is quite clear that the time consumed by any elementary computational operation can be bounded above (by the order of magnitude) by the total record length of its operands in binary encoding.*

**Definition 2.5** (running time)**.** The *running time* of an algorithm is the number of arithmetical and logical operations, as well as of operations of extracting the values of array elements. The *bit running time* of an algorithm is the number of bit operations estimated by taking into account the number of different computational operations (of all types) performed in the algorithm, and of the maximum possible record length of their operands.

**Parameters and objects of the model**

***Jobs, machines, the original transportation network:***
$n$ — the number of jobs/objects;
$m$ — the number of machines;
$\{J_1, \ldots, J_n\}$ — the set of jobs;
$\{M_1, \ldots, M_m\}$ — the set of machines;
$O_j^k$ $(k \in [m],\ j \in [n])$ — the operation of job $J_j$ on machine $M_k$;
$G = (V, E)$ — the original transportation network;
$v_0 \in V$ — *the depot* (the place of the starting and of the completing stays of each machine);
$\rho'(e)$ — *the length of an edge* $e \in E$; we assume that $\rho'(e) \in \mathbb{N}$ for all $e \in E$.

**Definition 2.6** (generalized operations)**.** By *generalized operations* we call both the machine operations on the objects, and the operations of their traveling between objects situated at **different** nodes of the transportation network.

**Definition 2.7** (related operations)**.** Two generalized operations are called *related*, if they are to be performed by the same machine or on the same object.

**Definition 2.8** (active/intermediate nodes)**.** The depot-node, as well as all nodes of the original transportation network containing objects are called *active nodes*. Thus, those and only those nodes are active, in which machines make stays. Nodes different from the depot are called *intermediate*.

***The reduced transportation network:***
$G^* = (V^*, E^*)$ denotes the *reduced transportation network* — a *complete graph* containing only active nodes of network $G$;
$V^* = \{v_1, \dots, v_g\}$ is the set of nodes of $G^*$ in their *natural numeration* (see Definition 2.9);
$g = |V^*|$ is the number of active nodes of network $G$;
$n_\nu$ ($\nu \in [g]$) is the number of objects at node $v_\nu \in V^*$;
$v_{\nu^*} \in V^*$ is the *depot* ($\nu^*$ is its *natural number*);
$E^*$ is the complete set of edges of network $G^*$;
$\rho(e)$ is the *length* of edge $e = (v_i, v_j) \in E^*$ ($i \neq j$) equal to the **length of the shortest path** between the corresponding nodes of network $G$ in any of the two directions; once due to the above agreement the length of any edge from $G$ belongs to $\mathbb{N}$, all values of $\{\rho(e)\}$ are also in $\mathbb{N}$ (i.e., each edge length is an integer number $\geq 1$; this fact is essentially used later in Lemma 3.4 while obtaining the upper bound on the number of stays of any machine in any optimal schedule); furthermore, it can be easily seen that function $\rho(e)$ meets *metric properties*;
$\rho_{\max} = \max\{\rho(v_i, v_j) \,|\, i, j \in [g],\ i \neq j\}$ is the maximum edge length in network $G^*$;
$|I|$ is the input length (in the compact encoding) of an instance $I$ of the $ROS^*$-$UET$ problem;
$\underline{|I|} \doteq \sum_{e \in E^*} \log \rho(e) + \log m + \log n$ is a lower bound on the input length (in the compact encoding) of a given instance $I$;
$\rho(R)$ the length of a route $R$ in network $G^*$;
$H^*$ is a hamiltonian cycle of minimum length in network $G^*$;
$\bar{C}$ is a lower bound on the optimum for a given instance of problem $ROS^*$-$UET$ defined by the formula

$$\bar{C} = \rho(H^*) + n. \tag{1}$$

**Definition 2.9** (natural numeration of nodes and objects, function $Loc(j)$)**.** A numeration of nodes **in nonincreasing order** of the number of objects located at a node ($n_1 \geq \dots \geq n_g$) will be called *natural*. A numeration of objects **in nonincreasing order** of the natural indices of the nodes containing those objects will be also called *natural*.
$N_\nu \doteq \sum_{i \in [\nu]} n_i$ ($\nu \in [g]$); $N_0 = 0$; $N_g = n$;
$\mathcal{J}(\nu) \doteq [N_{\nu-1} + 1, N_\nu]_{\mathbb{Z}}$ is the interval of natural indices of the objects located at node $v_\nu$;
$\mathcal{O}(\nu) = \{O_j^k \,|\, k \in [m],\ j \in \mathcal{J}(\nu)\}$ is the set of operations on objects at node $v_\nu$ ($\nu \in [g]$);
$Loc(j) = \min\{\nu \in [g] \,|\, N_\nu \geq j\}$ is the function determining the natural index of a node containing the object with natural index $j \in [n]$.

It should be noted that there is no need in computing the function $Loc(j)$ in the algorithm, because the node containing object $j$ is known in advance. Furthermore, we need not maintain and calculate the global numeration of the objects (and jobs), since their local numeration at each node is sufficient for the algorithm. Yet in the latter case we would have to denote each job by a pair of indices $(\nu, j)$ (for instance, in notation of operations), which is not convenient. Thus, the function $Loc(j)$ is introduced mostly for the sake of the convenience of notation in our paper.

**Parameters of schedules**

**Definition 2.10** (stays)**.** In any schedule $S$ at any time from $[0, F_{\max}(S)]$ every machine is either traveling (through edges and nodes of the transportation network), or performing a *stay* at a node. A stay of a machine at a node corresponds to the **maximal by inclusion** time interval within which the machine is located at that node. At any time from that interval, the machine either performs one of the operations located at that node or *is idle*. Stays of a machine, different from its first and last stays (being performed at the depot), are called *intermediate*. Given known a schedule of operations, we say that *a stay is nonempty*, if the machine performs at least one operation in it. Otherwise, the stay is called *empty*. The stay at which an operation $O'$ is performed, is called a *personal stay* of that operation.

Since the number of stays is essential for estimating the running time of the algorithm, it should be specified at which cases staying of a machine at a node is considered as a "stay" (empty or nonempty), and at which ones this is not the case. This question is resolved in our paper in a simple way, by switching from $ROS$ to the $ROS^*$-problem on network $G^*$ containing only active nodes. Moreover, we may also assume that in network $G^*$ every machine has only nonempty intermediate stays, moving between each pair of consequent stays (located at different nodes of network $G^*$) through the single edge connecting those nodes. At that, for each machine we admit the existence of an empty starting and/or finishing stay (at the depot), provided that such stays must have zero durations.

**Definition 2.11** (special nodes/jobs/operations/stays)**.** Node $v_\nu$ of network $G^*$ is called *special* (*nonspecial*), if $n_\nu < m$ ($n_\nu \geq m$). We denote by: $g_{ns}$ — the number of *nonspecial* nodes; $SN \doteq [g_{ns}+1, g]_{\mathbb{Z}}$ — the set of indices of special nodes; $g_{sn} \doteq |SN|$ — the number of *special nodes*; $g = g_{ns} + g_{sn}$. The jobs (as well as their operations) located at a special node are called *special jobs* (*operations*). The stays at special nodes are called *special stays*.

***Machine routes (the basic form):***
$R_k = \langle T_1^k, \dots, T_{s_k}^k \rangle$ is a *route* of machine $M_k$, or a chronological sequence of its stays; $s_k$ is the number of stays of machine $M_k$;
$T_t^k = (\nu_t^k;\ b_t^k, e_t^k)$ $(k \in [m],\ t \in [s_k])$ contains the complete information on the $t$th stay of machine $M_k$, where $t$ denotes the *machine index* of the stay, $\nu_t^k$ denotes the natural index of the node, where the $t$th stay of machine $M_k$ is performed, $b_t^k, e_t^k \in \mathbb{R}^+$ denote the starting and completion times of the stay;
$\mathcal{R} = \{R_k \,|\, k \in [m]\}$ is the complete package of machine routes;
$s_{k,\nu}$ denotes the number of stays of machine $M_k$ at node $v_\nu$.

***Machine routes (the alternative form):***
$\widehat{T} = \langle \widehat{T}(1), \dots, \widehat{T}(s) \rangle$ is *the total sequence of stays* $\{\widehat{T}(q)\}$ indexed in the order of lexicographical increasing of pairs $(b(q), \mu(q))$:

$$(b(1), \mu(1)) <_{\text{lex}} (b(2), \mu(2)) <_{\text{lex}} \cdots <_{\text{lex}} (b(s), \mu(s)); \tag{2}$$

$\widehat{T}(q) = (\mu(q), \nu(q), b(q), e(q))$ contains the complete information on the $q$th stay in sequence $\widehat{T}$: $\mu(q)$ is the machine index, $\nu(q)$ is the node index, $b(q), e(q)$ are the starting and the completion times of stay $q$; $q \in [s]$ is called an *absolute stay index*; $s$ is the total number of stays.
A stay characterized by the two-dimensional vector $T(q) = (\mu(q), \nu(q))$ will be called a *semi-stay*. The *total sequence of semi-stays* is a sequence $T = \langle T(1), \dots, T(s) \rangle$

defined by relations (2) (if all moments $\{b(q)\}$ are known) or defining such relations (when $\{b(q)\}$ are unknown).

Note that these two forms of routes (the basic form and the alternative one) are **informa-tively equivalent**, since each of the two can be completely determined from the other.

***An absolute job schedule, a complete schedule:***
$SJ = \{C(O_j^k) \,|\, k \in [m],\ j \in [n]\}$ is an *absolute job schedule*, where $C(O_j^k) \in [1, \infty)$ is the completion time of operation $O_j^k$;
$S = \langle \mathcal{R}, SJ \rangle$ is a *complete schedule*;
$C(S) = \max_{k \in [m]} e_{s_k}^k$ is the length of schedule $S$.

***A relative job schedule:***
$RS(\nu) \doteq \{LC(O_j^k) \,|\, k \in [m],\ j \in \mathcal{J}(\nu)\}$ is a relative job schedule at node $v_\nu$, where $LC(O_j^k) = (LC_1(O_j^k), LC_2(O_j^k))$ are the *local coordinates* of operation $O_j^k$;
$LC_1(O_j^k) \in [s_{k,\nu}]$ is the *local index* (see Definition 2.14) of the *personal stay* (see Definition 2.10) of operation $O_j^k$;
$LC_2(O_j^k) \in [n_\nu + \Delta - 1]$ is the value of the *shift* of the **completion time** of operation $O_j^k$ with respect to the **starting time** of its *personal stay* (for $\Delta \in [m]$).
The absolute completion time of operation $O_j^k$ can be computed by the formula

$$C(O_j^k) = \tau(O_j^k) + LC_2(O_j^k), \tag{3}$$

when the starting time $\tau(O_j^k)$ of its personal stay becomes known.

**Definition 2.12** (non-simultaneity)**.** Two operations are called *non-simultaneous*, if in a given (either absolute or relative) schedule the time intervals of their processing are intersecting at at most one point (being boundary for both intervals).

**Definition 2.13** (active schedule)**.** A schedule is called *active*, if no starting/completion time of a generalized operation can decreased without changing the mutual order of processing some pair of related operations.

It follows from the above definition that in any active schedule: (a) any empty stay has zero length; (b) any nonempty stay completes synchronously with the completion time of some operation; (c) the movement of each machine between any two consequent stays (which have to be performed at different nodes of the network) is carried out along the shortest path in the shortest time.

***Determinative parameters of a schedule:***
Given a complete schedule $S$, we define five its *determinative parameters*:
$\Delta_{[S]}$ is a parameter conformed with schedule length by the equality: $\Delta_{[S]} \doteq C(S) - \bar{C} + 1$, where $\bar{C}$ is the lower bound on the optimum defined in (1);
$s_{[S]}$ is the total number of semi-stays in schedule $S$;
$T_{[S]} = \langle T_{[S]}(1), \ldots, T_{[S]}(s_{[S]}) \rangle$ is the *total sequence of stays* of schedule $S$ defined by relations (2); given $T_{[S]}$, we can determine the increasing sequence $\mathcal{K}_{[S]} = \langle \bar{q}(1), \ldots, \bar{q}(\bar{s}) \rangle$ of *absolute indices* of special stays (where $\bar{s}$ is the number of special stays; their numeration by indices $t \in [\bar{s}]$ will be called *relative*).
It should be noted that for any complete schedule $S$ sequence $T_{[S]}$ is **uniquely** defined by relations (2), since no machine may have two and more simultaneously starting stays;
$D_{[S]} :\ [\bar{s} + 1] \to [0, 2m]$ is the *displacement* function defined for each special stay

(with a relative index $t \in [2, \bar{s}]_{\mathbb{Z}}$) by the amount $\delta(t) \doteq b(\bar{q}(t)) - b(\bar{q}(t-1))$ that its starting point is *shifted* with respect to the starting point of the previous special stay:

$$D_{[S]}(t) = \begin{cases} \delta(t), & \text{if } \delta(t) < 2m; \\ 2m, & \text{if } \delta(t) \geq 2m; \end{cases} \tag{4}$$

we also set $D_{[S]}(1) = D_{[S]}(\bar{s}+1) = 2m$;
$RSS_{[S]} = \{RS(\nu) \,|\, \nu \in SN\}$ is a *relative schedule* of jobs located at special nodes.

**Notions and objects used in algorithm $\mathcal{A}$**

***The Big Cycle*** is the *main part* of algorithm $\mathcal{A}$; it represents a cycle enumerating a finite set of possible variants of *pre-schedule* $P$. At the iterations of Big Cycle we are searching for a variant of pre-schedule extendable up to a feasible complete schedule $S$. The first such variant found provides an optimal schedule.

Next, we summarize all definitions of different numerations of stays.

**Definition 2.14** (numerations of stays)**.** A numeration of stays by indices $1, 2, \ldots$ consensual with relations (2) and defined for the sets of: all stays of a schedule/all special stays/all stays of machine $M_k$/all stays of machine $M_k$ at node $v_\nu$ — is called an *absolute/relative/machine/local numeration*; the index of a stay is called, respectively, an *absolute/relative/machine/local index*.

***Pre-schedule*** $P$

**Definition 2.15.** A combination (maybe, unrealizable) $P = \langle \Delta, s, T, D, RSS \rangle$ of possible values of determinative parameters $\langle \Delta_{[S]},\ s_{[S]}, T_{[S]}, D_{[S]}, RSS_{[S]} \rangle$ of a complete feasible schedule $S$ will be called a *pre-schedule*. Further, $B(X)$ denotes the number of variants of component $X \in \{\Delta, s, T, D, RSS\}$ of a pre-schedule $P$, to be enumerated in algorithm $\mathcal{A}$.

***Feasible values of components and the number of variants of pre-schedule $P$. Additional information extractable from $P$***

***Component $\boldsymbol{\Delta}$***. Based on the definition of the determining parameter $\Delta_{[S]}$ and on Property P2 of any optimal schedule $S$ (being proved in Lemma 3.3, page 60), we bound the values of the component $\Delta$ by the interval $[m]$. In turn, any value of $\Delta$ defines an upper bound on the length of schedule $S$ under construction:

$$C(S) \leq L(\Delta) \doteq \bar{C} + \Delta - 1. \tag{5}$$

***Component $s$***. Based on Property P5 of the optimal schedule $S^*$ (the existence of which is proved in Lemma 3.3), we bound the values of the component $s$ by the interval $[m(g+1), m(m+2g-2)]_{\mathbb{Z}}$.

***The sequence of semi-stays $\boldsymbol{T = \langle T(1), \ldots, T(s) \rangle}$*** is defined by a sequence consisting of $s$ pairs $T(q) = (\mu(q), \nu(q)) \in [m] \times [g]$.

***Additional information extractable from $T$:***
$s_k$, the number of stays of machine $M_k$ ($k \in [m]$);
$s_{k,\nu}$, the number of stays of machine $M_k$ at node $v_\nu$;
$s(\nu) = \sum_{k \in [m]} s_{k,\nu}$, the total number of stays at node $v_\nu$; $s(\nu) = |\mathcal{O}(\nu)|$;
$\bar{s}$, the number of *special stays*;
$\gamma(k, t')$, the function determining the *absolute index* of a stay of machine $M_k$ by its

*machine index* $t' \in [s_k]$;
$\kappa(t)$, the function determining the *machine index* of a special stay by its *relative index* $t$;
$\bar{q}(t)$, the function determining the *absolute index* of a special stay by its *relative index* $t$;
$RN(k, \nu; t'')$, the function determining the *relative index* of a special stay of machine $M_k$ at node $v_\nu$ ($\nu \in SN$) by its *local index* $t'' \in [s_{k,\nu}]$;
$\beta(k, \nu; t'')$, the function determining the *absolute index* of a stay of machine $M_k$ at node $v_\nu$ ($\nu \in [g]$) by its *local index* $t'' \in [s_{k,\nu}]$;
$R_k^T = \langle \gamma(k,t) \,|\, t = 1, 2, \ldots, s_k \rangle$, a *$T$-route of machine* $M_k$ (or the increasing sequence of *absolute indices* of stays of machine $M_k$);
$\rho(R_k^T) = \sum_{t=2}^{s_k} \rho(\nu(\gamma(k, t-1)), \nu(\gamma(k,t)))$, the length of route $R_k^T$;
$Q(k, \nu)$, the set of *absolute indices* of stays of machine $M_k$ at node $v_\nu$;
$\mathcal{K} = \langle \bar{q}(1), \ldots, \bar{q}(\bar{s}) \rangle$, the increasing subsequence of absolute indices of *special stays*.

**Definition 2.16** (ANS- and semi-special nodes)**.** Node $v_\nu$ of network $G^*$ will be called an *absolutely-non-special* node (or *ANS-node*, for short), if for a given pre-schedule $P = \langle \Delta, s, T, D, RSS \rangle$, the inequality $n_\nu \geq 2s(\nu)$ holds. If $m \leq n_\nu < 2s(\nu)$, the node is called *semi-special*.

***Feasible values of the component $T$ and the number of its variants***
The sequence of semi-stays $T$ should meet the following requirements:
(a) any two consequent stays of machine $M_k$ should be located **at different nodes**: $\nu(\gamma(k,t')) \neq \nu(\gamma(k,t'+1)),\ k \in [m],\ t' \in [s_k - 1]$;
(b) the values of $s_{k,\nu^*}$ ($k \in [m]$) should be in the interval $[2, n_{\nu^*} + 2]_{\mathbb{Z}}$, while the values of other amounts $s_{k,\nu}$ ($k \in [m],\ \nu \in [g] \setminus \{\nu^*\}$) should be in the intervals $[n_\nu]$ (in particular, all $s_{k,\nu} > 0$, which means that each machine should visit **each node** of network $G^*$ at least once);
(c) $s_{k,\nu} \leq g + (\Delta - 1)/2$ ($k \in [m],\ \nu \in [g]$);
(d) the subset $\{T(q) \,|\, q \in [m]\}$ of the first $m$ stays in $T$ should be compound of the **first stays** of all machines $M_1, \ldots, M_m$ (being performed at the depot and starting at time 0); they are marked by pairs $(\mu(q), \nu(q)) := (q, \nu^*)$; it should be noted that in a feasible schedule no stay at a node different from the depot may start at time 0, since the passage of a machine from the depot to another node requires positive time; by that reason (as well as due to property (a)), no non-first stay of any machine at the depot can start at time 0 either; as a result, **the first stays of all machines (and only them)** should start at time 0; the subset of the first $m$ stays of sequence $T$ should be entirely compound of the first stays of all machines (in the *semi-schedule*, which is to be found in problem $ILP(P)$ in Section 3.5, these stays will receive zero starting times: $b(q) := 0,\ q \in [m]$);
(e) $\nu(\gamma(k,1)) = \nu(\gamma(k,s_k)) = \nu^*,\ k \in [m]$, which means that the first and the last stays of each machine $M_k$ should be performed at the depot.

Let us estimate the number of variants of component $T$: $B(T) = (mg)^s \leq 2^{O(m(m+g)\log mg)}$.

***Function $D$ (displacement) and additional information extractable from it***

Function $D:\ [\bar{s}+1] \to [0, 2m]_{\mathbb{Z}}$ provides a semi-rigid time skeleton for the future schedule $S$, compound of special stays. "Semi-rigid", because only starting times of some pairs of stays are tied hard, while the completion times of stays remain unspecified. The distance between starting times of "neighboring" special stays

(having "neighboring" indices $t-1$ and $t$ in the relative numeration of stays) is specified in schedule $S$ by the relations

$$b(\bar{q}(t)) - b(\bar{q}(t-1)) \begin{cases} = D(t), & \text{if } D(t) < 2m; \\ \geq D(t), & \text{if } D(t) = 2m \end{cases}$$

for $t = 2, \ldots, \bar{s}$. We also set $D(1) = D(\bar{s}+1) = 2m$. Let us estimate **the number of variants of component** $D$: $B(D) \leq (2m+1)^{s-1} \leq 2^{O(m(m+g)\log m)}$.

**Definition 2.17** (a segment of the sequence of special stays)**.** Function $D$ divides the interval $[\bar{s}]$ (of relative indices of special stays) into *segments*, divisible from each other by values $D(t) = 2m$. Let $\mathcal{T}^* \doteq \langle t_1^*, t_2^*, \ldots, t_{\hat{s}}^* \rangle$ be the increasing sequence of indices $t \in [\bar{s}+1]$ for which $D(t) = 2m$. (The sequence is nonempty: at least, we have $D(1) = 2m$, implying $t_1^* = 1$.) Then the $\xi$th *segment* is defined as the set $\{t \in [\bar{s}] \,|\, t_\xi^* \leq t < t_{\xi+1}^*\}$.

**Definition 2.18** (a personal segment and segment coordinates of a stay)**.** Segment $S(t)$ containing the special stay with a relative index $t \in [\bar{s}]$ will be called a *personal segment* of that stay. Let us define *segment coordinates* for each special stay $t$ as a pair $(SC1(t), SC2(t))$, where $SC1(t)$ denotes the relative index of the first stay of segment $S(t)$, and $SC2(t)$ shows the shift of the starting time of stay $t$ with respect to the starting time of stay $SC1(t)$. The segment coordinates of stay $t$ can be computed by the recurrent formulas:

$$\begin{array}{lll} SC1(t) := t, & SC2(t) := 0, & \text{if } D(t) = 2m; \\ SC1(t) := SC1(t-1), & SC2(t) := SC2(t-1) + D(t), & \text{if } D(t) < 2m \end{array}$$

(as can be easily seen, they are integral).

***A relative schedule of special jobs (RSS):*** $RSS$ is a family $\{RS(\nu) \,|\, \nu \in SN\}$ of relative schedules at special nodes, where $SN$ is the index set of special nodes, and $RS(\nu)$ (a relative schedule of jobs located at node $v_\nu$) is defined on page 50 as a family $\{(LC_1(O_j^k),\ LC_2(O_j^k)) \,|\, k \in [m],\ j \in \mathcal{J}(\nu)\}$ of pairs of *local coordinates* for all operations at node $v_\nu$. The absolute completion time of each special operation will be computed by the formula

$$C(k,j) = b(\beta(k, Loc(j); LC_1(O_j^k))) + LC_2(O_j^k) \tag{6}$$

after getting a solution of the $ILP'(P)$-problem, when the starting and the completion times $(\{b(q), e(q) \mid q \in [s]\})$ of all stays become known.

***Additional information extractable from RSS:***
Having $RSS$, we can compute *durations of special stays*:

$$A(P,t) := \max_{O_j^k \in \mathcal{O}[t]} LC_2(O_j^k),\ t \in [\bar{s}] \tag{7}$$

(where $\mathcal{O}[t]$ denotes the set of operations performed at stay $t \in [\bar{s}]$), as well as

***Segment coordinates*** $(SC_1(O_j^k), SC_2(O_j^k))$ ***of special operations***, where $SC_1(O_j^k) \doteq SC1(RN(k, Loc(j); LC_1(O_j^k)))$ is specified by the segment coordinate $SC1$ of the *personal stay* of operation $O_j^k$, and $SC_2(O_j^k) \doteq SC2(RN(k, Loc(j); LC_1(O_j^k))) + LC_2(O_j^k)$ specifies the *shift* of the **completion time** of operation $O_j^k$ with respect to the **starting time** of stay $SC_1(O_j^k)$. These coordinates can be computed in the algorithm synchronously with specifying a pre-schedule $P$, and allow one to check $RSS$ for compliance with all non-simultaneity requirements to

related special operations. In Lemma 3.7 (Section 3.4) a criterion of non-simultaneity of two special operations is proved. It runs that two special operations are non-simultaneous, if and only if they mismatch as points in the two-dimensional space of their segment coordinates, which allows one to check this property efficiently.

***Feasible values of the component RSS and an upper bound on the number of its variants***

Local coordinates $LC_1(O_j^k)$ and $LC_2(O_j^k)$ of special operations $\{O_j^k\}$ receive values from the integral intervals $[s_{k,\nu}] \subseteq [m]$ and $[n_\nu + \Delta - 1] \subseteq [2m-2]$, respectively, with $\nu = Loc(j)$ (subject to the restrictions set to the amounts $s_{k,\nu}$ in requirement (b) to component $T$). A relative schedule $RSS$ is called feasible, if it meets the requirement of non-simultaneity: any two *related* special operations in a feasible schedule should be *non-simultaneous* (see Definitions 2.7 and 2.12).

Subject to the upper bound $g_{sn}m^2$ on the number of special operations (where $g_{sn}$ is the number of special nodes in network $G^*$), the overall **number of variants of component** $RSS$ can be estimated as $B(RSS) \leq (2m^2)^{g_{sn}m^2} \leq 2^{O(g_{sn}m^2 \log m)}$.

**Definition 2.19** (a feasible pre-schedule)**.** Pre-schedule $P = \langle \Delta, s, T, D, RSS \rangle$ is *feasible*, if all above mentioned requirements to its components are met.

Given $\Delta' \in [m]$, $\mathcal{P}(\Delta')$ denotes the set of feasible pre-schedules $P = \langle \Delta, s, T, D, RSS \rangle$ with values $\Delta \leq \Delta'$; $\mathcal{P}$ will denote the set $\cup_{\Delta' \in [m]} \mathcal{P}(\Delta')$ of all feasible pre-schedules for a given instance of problem $ROS^*$-$UET$.

***Pre-schedule/semi-schedule/complete schedule:***

In our algorithm, we distinguish three stages of readiness of a schedule: *a pre-schedule*, *a semi-schedule*, and *a complete schedule*. To obtain a pre-schedule, we specify the values of so called *determinative parameters* of a schedule, whose ranges of values can be restricted by functions independent of the number of objects $n$. Since the number of determinative parameters is also bounded by a function independent of $n$, this implies that even the complete enumeration of all combinations of values of those parameters requires time bounded by a function depending on parameters $m$ and $g$ only.

Given a pre-schedule $P$, by solving the problem $ILP'(P)$ (formulated in Section 3.4), we find a *semi-schedule*. It is characterized by the property that **the whole information on machine routes** becomes known. (Thus, one of the two components of a complete schedule becomes completely defined.) A local schedule of jobs at each special node is also known.

**Definition 2.20** (an extension of a schedule)**.** Suppose, we are given two partial schedules: $S'$ and $S''$. We say that schedule $S''$ is *an extension* of schedule $S'$, if the list of parameters of schedule $S''$ with known values includes that of schedule $S'$.

For instance, given a pre-schedule, we construct a semi-schedule which is its extension. The *complete schedule* being built next is an extension of the semi-schedule.

## 3. Preliminary results and analysis of properties of optimal schedules

In the first lemma presented in this section, we establish not only the fact of the existence of an optimal schedule for any given instance of the $ROS$-problem,

but also the existence of such a schedule with certain properties. The next sub-section contains a polynomial-time reduction of the $ROS$-problem to the $ROS^*$-problem on the transportation network $G^*$ possessing a number of nice properties. Subsequently, we pass over to constructing and analysis of an algorithm for solving the $ROS^*$-$UET$ problem. Next, in Sub-section 3.2, a parameterized approximation algorithm for solving the $ROS^*$-$UET$ problem is presented and its absolute performance guarantee is established, which is further implemented for deriving a bound on the running time of the exact algorithm. In Sub-section 3.3, we prove the existence of an optimal schedule of problem $ROS^*$-$UET$ with a bunch of such properties that enable us to restrict the number of variants of pre-schedules under enumeration by a function depending only on parameters $m$ (the number of machines) and $g$ (the number of nodes of network $G^*$). In the next sub-section (3.4) two integer linear programming problems ($ILP(P)$ and $ILP'(P)$) are considered, being used in the algorithm of extending a given *pre-schedule* $P$ up to a feasible *semi-schedule*. A close relation between these two problems is established. Next, in Sub-section 3.5, two polynomial-time procedures of extending a given semi-schedule (obtained as a solution of the $ILP'(P)$-problem) up to a complete feasible schedule are presented. Lemma 3.10 proved in Sub-section 3.6 is a key result used for justification of our main algorithm.

3.1. **The lemma on the existence of an integral optimal solution of the $ROS$-problem.** Since in the $ROS$-problem it is required that each machine has to walk around all objects (performing on each object exactly one operation), and since the route of each machine starts and ends at the depot, it has, thereby, to walk around all *active* nodes of network $G$. It follows that for the existence of a feasible solution it is required that the sub-network corresponding to the set of all active nodes must be connected. Since, next, it is well known that the problem of verifying the connectivity of any given subset of vertices of a graph is polynomially solvable, we may further assume (without loss of generality of the problem under solution) that in any problem instance specified at the input of the $ROS$-problem the network is connected. That the connectivity of the network is also a sufficient condition for the existence of an optimal solution of the $ROS$-problem, we get known from the following

**Lemma 3.1.** *For any instance of the ROS-problem, there exists an optimal schedule with the following properties: (a) all essential events (such as the beginning and the completion of an operation, the beginning and ending of a machine movement, the beginning and ending of a machine stay at a node, the starting and ending of the whole project) happen at* ***integral points in time****; (b) the ending time of each stay of a machine (with the exception of its last stay) coincides with the starting time of its movement to a* ***different node of the network****; the ending time of its last stay coincides with the completion time of some generalized operation of that machine; (c) every time moment of starting a movement of a machine coincides with either the completion time of an operation processed by that machine, or with time* 0*; (d) the movement of a machine from one active node to another one is carried out* ***along the shortest way in the shortest time****.*

*Proof.* To begin with, we note that the fact of existing a non-empty set of **feasible schedules** for any given instance of the $ROS$-problem is evident. (To obtain such a schedule, it is sufficient to provide for each machine, consequently, a possibility

to perform all its operations in an arbitrary order.) For an arbitrary given feasible schedule $S$, we define a *precedence graph* for the set of all *generalized operations* (see Definition 2.6), being performed by machines according to schedule $S$. In view of the *non-simultaneity requirements* to *related operations* (see Definitions 2.12 and 2.7), in any feasible schedule $S$ between any two related operations $\{o', o''\}$ one of the two precedence relations may hold: either $o' \prec o''$, or $o'' \prec o'$, where $o' \prec o''$ means: "$o''$ starts not earlier then $o'$ is completed". The family of such relations can be modeled by a network model of the type "jobs-arcs" (or by a *directed graph* $\widetilde{G}(S) = (\widetilde{V}, \tilde{A})$ with weighted arcs), in which to each generalized operation $o'$ correspond two vertices ($b(o')$ and $e(o')$), representing the *events* of starting and completion of the operation, and an arc $(b(o'), e(o'))$ with length equal to the duration of the operation. If $\{o', o''\}$ are two consecutive operations on objects, which are to be performed by machine $M_k$ at different nodes $(v_{\nu'}, v_{\nu''})$ of network $G$, then between the corresponding vertices in the precedence graph should be the operation of the machine movement from node $v_{\nu'}$ to node $v_{\nu''}$. The length of such a generalized operation (and of the corresponding arc in graph $\widetilde{G}(S)$) is equal to the *distance* between the nodes $v_{\nu'}$ and $v_{\nu''}$, i.e., to the length of the shortest simple path connecting these nodes in $G$. Beside that, an operation of a machine movement from $v_{\nu'}$ to $v_{\nu''}$ is added in two more cases: 1) when $v_{\nu'}$ is the depot, while the first operation of the machine is located at another node ($v_{\nu''}$), and 2) when the last operation of a machine is located at node $v_{\nu'} \neq v^* = v_{\nu''}$. And lastly, it is clear that when two operations of a machine are being processed consecutively (in schedule $S$) at the same node of $G$, then we need to add no "movement operation" of the machine between them (as well as no superfluous arc in graph $\widetilde{G}(S)$).

Let now two generalized related operations ($o'$ and $o''$) be performed in schedule $S$ **directly one after another** in order $o' \prec o''$. Then, to specify this order of their processing, we add an arc $(e(o'), b(o''))$ of zero length to graph $\widetilde{G}(S)$. Beside that, for each machine $M_k$ we add to graph $\widetilde{G}(S)$ two vertices-events: $\alpha_k$ and $\omega_k$, denoting the start of the very first, and the end of the very last stays of machine $M_k$ (both stays being performed at the depot). Vertex $\alpha_k$ is connected by a zero-length arc with the starting event of the first generalized operation of that machine, while the completion event of its very last generalized operation is connected by a similar arc with vertex $\omega_k$. And finally, we add to graph $\widetilde{G}(S)$ vertices $v^*$ and $v^{**}$ denoting *the start* and *the completion of the whole project*, and add also zero-length arcs from $v^*$ to every vertex $\alpha_k$, and from each vertex $\omega_k$ to vertex $v^{**}$, which completes the construction of graph $\widetilde{G}(S)$.

As can be easily seen, graph $\widetilde{G}(S)$ is a *reduction graph* of the partial order $\prec \doteq \prec_S$, defined (for a given schedule $S$) on the set of generalized operations and on the set of events generated by those operations. We should also note that at each point in time each machine is either making a *stay* at a node (see Definition 2.10), or is traveling between two successive stays. Henceforth, each starting event of a machine movement from one node to another coincides with the completion event of the previous stay of the machine, while the ending event of a machine movement coincides with the starting event of the next stay of the machine.

In so defined network $\widetilde{G}(S)$, there exists a tight lower bound ($\tilde{t}$) on the moment of any event $v \in \widetilde{V}$ (a so called *earliest moment* of the event), realizable on the longest (*critical*) path in graph $\widetilde{G}(S)$ from vertex $v^*$ to vertex $v$. As is easily seen,

there are no positive-length cycles in $\widetilde{G}(S)$ (since the existence of such a cycle would contradict the feasibility of the given schedule $S$). Thus, a sufficient condition for the existence of a feasible *active schedule* $S_{\prec}$ for the set of all events in graph $\widetilde{G}(S)$ is satisfied. This schedule is also feasible for the set of operations in the *ROS*-problem; furthermore, it minimizes the occurrence time $\tau(v)$ of **each event** $v \in \widetilde{G}(S)$ over all feasible schedules with a given partial order specified on each pair of related operations. At that, $\tau(v)$ is equal to the length of the critical path from vertex $v^*$ to vertex $v$. Since the length of each arc is integral, this implies the integrality of the occurrence time of each event in schedule $S_{\prec}$, including the completion event of the whole project. This means the integrality of the length of schedule $S_{\prec}$.

The active schedule $S_{\prec}$ has the minimum length among feasible schedules with a given order $\prec$. Having enumerated all variants of the order of performing related operations (the number of such variants is finite) and all corresponding schedules $S_{\prec}$, we can find a schedule $\widetilde{S}$, on which the global minimum of schedule length is attained. Property (b) of the schedule obtained follows from the definition of *stay*; properties (c) and (d) follow from the defining property of an active schedule and from the construction of graph $\widetilde{G}(S)$. □

The lemma proved above implies the following

**Corollary 3.1.** *In an algorithm of searching for an optimal solution of the ROS-problem, we may restrict ourselves by considering those schedules only possessing the properties listed in Lemma 3.1.*

3.2. **A reduction from the *ROS*-problem to the *ROS**-problem. An approximation algorithm for the *ROS**-*UET* problem.** The original transportation network is modeled by an ordinary graph $G$ with weighted edges, where the *edge weight* is defined as the **time** required for the movement of any machine (and in any of two possible directions) between the nodes connected by the edge. Since the whole job in this model is being performed in active nodes only of network $G$ (between which the machines move along the shortest routes, by Corollary 3.1), while all the remaining (not active) nodes of the network are used as "transit" nodes only, it makes sense to pass over to a *reduced* network $G^*$, containing *active nodes* only (see Definition 2.8). Thus, network $G^* = (V^*, E^*)$ represents a **complete ordinary graph**, whose set of vertices coincides with the set of active nodes of network $G$, while the weight $\rho(i, j)$ of each edge $(v_i, v_j) \in E^*$ $(i \neq j)$ is equal to the time of machine movement (along **the shortest path**) between the corresponding nodes of the original network $G$. As follows from the definition of the function $\rho(i, j)$, it meets all **metric properties** (from which only two will be used: *symmetry* and *transitiveness*). From now on, the *ROS*-problem on the reduced transportation network $G^*$ will be called a *ROS*-problem.*

**Proposition 3.1.** *The ROS-problem can be reduced to the ROS*-problem in polynomial time. The running time of the reduction is equal (by the order of magnitude) to the polynomial running time of the algorithm of searching for the bunch of shortest routes between all pairs of nodes from a given subset $V^* \subseteq V$ of nodes of the original network $G$.* □

In view of Proposition 3.1, to construct a parameterized algorithm for the exact solution of the *ROS-UET* problem, it is sufficient to design such an algorithm for

the *ROS*-UET* problem. We start the analysis of the latter by deriving a lower bound on its optimum.

**Proposition 3.2.** *The length of any feasible route of any machine $M_k$ ($k \in [m]$) (and as a corollary, the length of any feasible schedule) in the ROS*-UET problem cannot be less than $\bar{C} \doteq \rho(H^*) + n$, where $\rho(H^*)$ is the shortest hamiltonian cycle in network $G^*$, and $n$ is the total number of jobs.*

*Proof.* Indeed, once in any feasible schedule $S$ any machine has to get around all the nodes of the network locating jobs (while in the *ROS*-UET* problem, all nodes of the network contain jobs, except, maybe, the node-depot), and since any machine route must start and end at the depot, this means that each machine has to get around **all nodes** of network $G^*$. As follows from the completeness of graph $G^*$ and from the transitiveness of eights of its edges, the minimum-length route passing through all nodes of network $G^*$ is a **hamiltonian cycle** of the minimum weight (denoted by $H^*$), which provides the summand $\rho(H^*)$ of the lower bound. The addend $n$ is added to the bound, since each machine performs **all jobs**. □

As was said above, the main result of the paper concerns designing an algorithm of the **exact solution** of the *ROS*-UET* problem. However, we first present a quite simple approximation algorithm for its solution. (The result obtained will be used further in the construction of the exact algorithm.)

**Lemma 3.2.** *For any instance of the ROS*-UET problem there exists a feasible schedule $S$, the length of which is no greater than $\bar{C} + m - 1$ (where $\bar{C}$ is the lower bound on the optimum derived in Proposition 3.2).*

*Proof.* Let $H^*$ be the shortest hamiltonian cycle. We define schedule $S$ as follows. Number the machines $\{M_1, \ldots, M_m\}$ arbitrarily, and let them go in this order, one after another, using the same route $H^*$, but with the time lag of every other machine (with respect to the previous one) equal to 1. For each machine $M_k$ at each node $v_\nu$ we organize a stay of length $n_\nu$, while which the machine should perform all the jobs located at that node. At that, all machines perform the jobs in the same order (which is defined for machine $M_1$ at each node $v_\nu$ arbitrarily). Since each operation has unit length, the unit shift of the schedule of each machine $M_k$ ($k > 1$) with respect to the previous machine ($M_{k-1}$) guarantees that the non-simultaneity requirement is met for all operations of each job. Since the compliance with all other requirements is evident, schedule $S$ is feasible. Its length ($\bar{C} + m - 1$) is defined by the time when the last machine ($M_m$) returns to the depot. □

This simple lemma (and its proof) imply a mass of interesting corollaries that will be used in Section 3.3 while deriving properties of optimal schedules. As a one more direct corollary, we obtain a $(m, g)$-parameterized approximation algorithm $\mathcal{A}'$ for solving the problem with an absolute performance guarantee. For $m$ and $g$ bounded by constants and for the increasing $n$, this algorithm becomes asymptotically optimal, and its running time becomes linear on the length of the input in the compact encoding. Algorithm $\mathcal{A}'$ uses (as a subroutine for solving the TSP problem) the DP-algorithm designed in 1962 by Bellman (and independently, by Held and Karp).

**Theorem 3.1.** (Bellman, R., 1962), (Held, M., Karp, R.M., 1962) *The TSP problem with $n$ cities and an arbitrary matrix of distances can be solved in $O(2^n n^2)$ elementary computational operations (see Definition 2.4).*

**Theorem 3.2.** *For the ROS*-UET problem, there exists an approximation algorithm $\mathcal{A}'$ that solves it with an absolute performance guarantee not greater than $m-1$ and with bit running time $O(2^g g^2+mg)|I|$ (where $|I|$ is the length of problem input in compact encoding).*

*Proof.* Let us take schedule $S$ constructed in the proof of Lemma 3.2 as a desired one. (Thus, we may skip the checking of its feasibility.) It follows from Proposition 3.2 and Lemma 3.2 that the length of schedule $S$ deviates from the optimum by at most the amount $m-1$. It remains to show that the schedule can be computed within the announced time.

To begin with, we find a hamiltonian cycle $H^*$ in graph $G^*$ (which requires $O(2^g g^2)$ elementary computational operations, due to Theorem 3.1), after which we renumber the nodes of network $G^*$ by indices $\{1,\dots,g\}$ in the order of their passage by the cycle $H^*$ (with the starting and the ending points at the depot): $H^*=(v_1,v_2,\dots,v_g,v_1)$. Let us compute the amounts $\rho_\nu=\sum_{i=2}^{\nu}\rho(v_{i-1},v_i)$, $N'_\nu=\sum_{i\in[\nu]}n_i$ $(N'_0=0)$, $t'_\nu=N'_{\nu-1}+\rho_\nu$, $t''_\nu=N'_\nu+\rho_\nu$ for all $\nu\in[g]$.

Next, we renumber the jobs at node $v_\nu$ arbitrarily by indices from the interval $[N'_{\nu-1}+1,N'_\nu]_{\mathbb{Z}}$. Unlike the *natural job numeration* (see Definition 2.9), we will call it a *hamiltonian* one. Then in the desired schedule $S$, machine $M_k$ $(k\in[m])$ performs the jobs located at node $v_\nu$ in the order of their hamiltonian numeration in the time interval $[t'_\nu,t''_\nu]+k-1$. To uniquely define such schedule $S$, it is sufficient to specify only the intervals $[N'_{\nu-1}+1,N'_\nu]_{\mathbb{Z}}$ of the indices of jobs located at nodes $v_\nu$ $(\nu\in[g])$ and the time intervals $[t'_\nu,t''_\nu]$ $(\nu\in[g])$ of performing the operations of those jobs by machine $M_1$. (The time interval of any other machine $M_k$, $k>1$, can be obtained from that of $M_1$ by shifting it forward by $k-1$.) Subject to the encoding length of the amounts $\{N'_\nu,t'_\nu,t''_\nu\,|\,\nu\in[g]\}$ (which is not greater than $g(\log n+2\log(\bar{C}+m-1))\le O(g^2(\log n+\log\rho_{\max}+\log m))\le O(g^2|I|)$), we obtain the claimed bound on the bit running time of the whole algorithm: $\mathcal{T}(\mathcal{A}')\le O(2^g g^2)|I|$.

Actually, the algorithm is $g$-parameterized, since it is sufficient to specify in schedule $S$ the time intervals **for only one machine** $(M_1)$. (The interval of each next machine $M_k$ can be obtained by the corresponding shifts by $(k-1)$ time units.) Yet, if it is required to specify explicitly the time intervals for all machines, then we will have to add the summand $mg|I|$ to the bound on running time. □

Lemmas 3.1, 3.2 and Proposition 3.2 imply

**Corollary 3.2.** *The optimum of any instance of the ROS*-UET problem exists and can be represented in the form $C(S^*)=L(\Delta)\doteq\bar{C}+\Delta-1$, where $\Delta\in[m]$, and $\bar{C}$ is the lower bound on the optimum established in Proposition 3.2.* □

3.3. **Properties of an optimal solution of the *ROS*-UET* problem.** In this subsection we proof the existence (for any given instance of the *ROS*-UET* problem) of an optimal schedule with a certain bunch of properties enabling one to reduce significantly the enumeration of variants of pre-schedules (in the algorithm of computing the optimal schedule, Section 4), which thereby reduces the running time of the algorithm.

**Properties of optimal schedules**

**P1.** The schedule is **entirely integral**, which means the integrality of all its temporal parameters (including *local and segment coordinates* of operations, the

values of function $D$, *etc.*) and — the integrality of the occurrence times of all significant events (such as the starting and the completion of an operation, the start and the end of a machine movement through the edges of the network, the start and the end of a machine stay at a node of the network, the completion of the whole schedule).

**P2.** The schedule length can be represented in the form: $C(S) = L(\Delta) \doteq \bar{C} + \Delta - 1$, where $\Delta \in [m]$, and $\bar{C}$ is the lower bound on the optimum established in Proposition 3.2.

**P3.** The total idle time of each machine is no greater than $m - 1$.

**P4.** Any two successive stays of a machine are performed at **different** nodes.

**P5.** The number of stays of any machine cannot be less than $g+1$ and is not greater than $2g-1+\Delta'$, where $\Delta' = C(S) - \bar{C} \leq m-1$; the total number of stays belongs to the interval $[m(g+1), m(m+2g-2)]_{\mathbb{Z}}$.

**P6.** The number of stays of any machine at any node is not greater than $g + \Delta'/2$, where $\Delta' = C(S) - \bar{C} = \Delta - 1$.

**P7.** In the schedule, there are no *empty* intermediate stays (see Definition 2.10), while any empty starting or ending stay of a machine has got zero length.

**P8.** The number of stays ($s_{k,\nu}$) of each machine $M_k$ at any node $v_\nu$, $\nu \neq \nu^*$, is not greater than $n_\nu$, and at node $v_{\nu^*}$ is not greater than $n_{\nu^*} + 2$. At that, if $s_{k,\nu^*} = n_{\nu^*} + 2$, then the first and the last stays of machine $M_k$ are empty, and all its operations at the depot are distributed among $n_{\nu^*}$ intermediate stays (with local indices from $[2, n_{\nu^*} + 1]_{\mathbb{Z}}$).

**P9.** The first local coordinate ($LC_1(O_j^k)$, see page 50) of any special operation $O_j^k$ at the condition $s_{k,\nu} \leq n_\nu$ (where $\nu \doteq Loc(j)$) takes values from the interval $[n_\nu] \subseteq [m-1]$, and at the alternative condition $s_{k,\nu} \in \{n_\nu + 1, n_\nu + 2\}$ (which, when P8 holds, is possible only for $\nu = \nu^*$) — from the interval $[n_\nu + 1] \subseteq [m]$. The second local coordinate ($LC_2(O_j^k)$) takes values from the interval $[2m-2]$.

**Lemma 3.3.** *There exists an optimal schedule $S^*$ with properties P1-P9.*

To prove Lemma 3.3, we need the following result.

**Lemma 3.4.** *Let $G = (V, E)$ be a connected $g$-vertex graph with integer positive weights of edges ($\{\rho(e) \mid e \in E\}$), and let $R$ be a closed walk passing through each vertex at least once and containing $2g - 2 + k$ edges ($k \in \mathbb{Z}^+$). Then $R$ has weight $\rho(R) \geq \rho(R^*) + k$, where $\rho(R^*)$ is the minimum weight of a closed walk passing through all vertices of graph $G$.*

*Proof.* If $k = 0$, we are done. Let $k > 0$, and let $H = (V_H, E_H)$ be a multigraph consisting of all vertices and edges of the closed walk $R$. Thus, $V_H = V$, and $E_H$ includes an edge $\{u, v\}$ of $G$ that many times the edge is contained in $R$. As a result, $H$ has got $g$ vertices and $2g-2+k$ edges. It is connected and Eulerian, so, it contains a spanning tree $T$ consisting of $g-1$ edges. The "additional" $g-1+k \geq g$ edges of $H$ comprise a cycle $C$ whose removal from $H$ results in a connected eulerian multigraph with a less number of edges. We repeat this deletion procedure until we get a multigraph $H'$ with at most $g-1$ "additional" edges. Since it is still eulerian, it includes an eulerian tour $R'$ through all vertices of $G$, and thus having the weight

$\rho(R') \geq \rho(R^*)$. Since at least $k$ edges of $R$ have been deleted, the weight of $R$ is at least $\rho(R^*) + k$. □

**Corollary 3.3.** *If for some $z \in \mathbb{Z}^+$ and for a given feasible schedule $S$ for the $ROS^*$-$UET$ problem the length of a route $R_k^S$ of machine $M_k$ is not greater than $\rho(H^*) + z$, then the number of its edges does not exceed $2g - 2 + z$.*

*Proof.* Suppose the contrary, i.e., for a feasible schedule $S$ obtained for the $ROS^*$-$UET$ problem the number of edges of the route $R_k^S$ is equal to $2g - 2 + z'$ for some integer $z' > z$. Then, since in a complete graph $G^*$ with edge weights satisfying the metric properties the shortest route through all vertices is a hamiltonian cycle, by Lemma 3.4 we have the relations $\rho(R_k^S) \geq \rho(H^*) + z' > \rho(H^*) + z$ contradicting the conditions of our lemma. □

**The proof of Lemma 3.3:** Let us take the optimal schedule $\widetilde{S}$ with properties (a)-(d), the existence of which was proved in Lemma 3.1 (page 55). Then property $P1$ coincides with property (a). Property $P2$ holds by Corollary 3.2, from which we directly derive property $P3$. Property $P4$ is a corollary of property (b).

Since, in view of property $P2$, the length of the route $R_k^{\widetilde{S}}$ of each machine $M_k$ is not greater than $\rho(H^*) + \Delta - 1$ for some $\Delta \in [m]$, by Corollary 3.3, the number of edges of the route $R_k^{\widetilde{S}}$ is not greater than $2g - 3 + \Delta$. Since, by property $P4$, between any two edges of the route of machine $M_k$ (as well as prior to the first, and after the last edge) machine $M_k$ performs exactly one stay, the number of its stays in the route $\rho(R_k^{\widetilde{S}})$ does not exceed $2g - 2 + \Delta$, which implies the upper bound on the total number of stays. The lower bound ($s_k \geq g + 1$) on the number of stays of machine $M_k$ follows from the feasibility of schedule $\widetilde{S}$ (since each machine has to visit each node of network $G^*$, and should visit the depot at least twice). Thus, property $P5$ holds.

Let us prove property $P6$. To begin with, let us estimate the maximum possible number of stays at the depot. As is known, any machine $M_k$ has the first stay at the depot. After that, in the route of the machine one can observer the repetition: "non-depot", depot, "non-depot",..., depot, where each "depot" means exactly one stay (at the depot), while each "non-depot" means several (at least one) stays at nodes different from the depot. Henceforth, by property $P5$, we obtain the desired bound: $s_{k,\nu^*} \leq (s_k - 1)/2 + 1 = (s_k + 1)/2 \leq (2g - 1 + \Delta)/2 \leq g + \Delta'/2$. For other nodes (different from the depot) a stronger bound holds:

$$s_{k,\nu} \leq (s_k - 1)/2 \leq (2g - 3 + \Delta)/2 \leq g - 1 + \Delta'/2.$$

Let us prove $P7$. Suppose that in schedule $\widetilde{S}$ there is an empty intermediate stay $X$ at node $v_\nu$. Then by property $(c)$, the completion of stay $X$ coincides with either time 0, or the completion of some operation $O'$. The first case is impossible, since stay $X$ is **intermediate**, while at the completion of the **starting stay** the machine had to move to **another node** (by property $(b)$), which requires a non-zero time. In the second case, since stay $X$ is empty, operation $O'$ is, clearly, performed in **another stay** $Y$, at the completion of which the machine should move to **another node** (by property $(b)$). And again, due to the strict positiveness of the weights of all edges of network $G^*$, the time passing between the stays $X$ and $Y$ cannot be zero. Thus, the second case is also impossible, which completes the proof of property $P7$.

Property $P8$ is a direct consequence of property $P7$.

Let us prove $P9$. The bound on the first local coordinate of a special operation follows from property $P8$ and the definition of a special node. The belonging of the second local coordinate ($LC_2(O_j^k)$) of a special operation to the interval $[2m-2]$ follows from its integrality (by property $P1$), the definition of a special node ($n_\nu \leq m-1$), and from the bound on the total idle time of a machine (property $P3$). Thus, schedule $\widetilde{S}$ possesses all properties claimed in Lemma 3.3. □

**Proposition 3.3.** *Pre-schedule $P_{[S^*]} = \langle \Delta_{[S^*]}, s_{[S^*]}, T_{[S^*]}, D_{[S^*]}, RSS_{[S^*]} \rangle$ of schedule $S^*$ belongs to the set $\mathcal{P}(\Delta_{[S^*]})$ defined on page 54. At that, $\Delta_{[S^*]} \in [m]$.*

*Proof.* Inclusion $\Delta_{[S^*]} \in [m]$ follows from the definition of this determining parameter (presented on page 51) and from the property $P2$ of schedule $S^*$. Let us show that pre-schedule $P_{[S^*]}$ of schedule $S^*$ meets all restrictions imposed on pre-schedules from the set $\mathcal{P}(\Delta_{[S^*]})$.

Restrictions on the overall number of stays $s_{[S^*]}$ are met in view of property $P5$. Property (a) of the sequence $T_{[S^*]}$ follows from property $P4$, property (b) — from property $P8$, property (c) — from $P6$. Property (d) follows from the feasibility of schedule $S^*$ (as well as property (e)) and from ordering the stays in $T_{[S^*]}$ according to (2). Next, the correspondence of the function $D_{[S^*]}(t)$ (defined in (4)) to the definition of component $D$ of a pre-schedule $P \in \mathcal{P}(\Delta_{[S^*]})$ can be easily checked.The correspondence of the relative schedule ($RSS_{[S^*]}$) of special jobs to the restrictions on the values of their local coordinates (imposed on page 50) follows from property $P9$. Proposition 3.3 is proved. □

3.4. **Two problems on constructing a feasible semi-schedule.** Denoting by variables $x(q)$, $y(q)$ unknown moments $b(q), e(q)$ of the beginning and the ending of stays $T(q)$ ($q \in [s]$), at each iteration of the Big Cycle of algorithm $\mathcal{A}$ we solve the problem $ILP(P)$ of constructing a feasible semi-schedule $S$ extending a given pre-schedule $P = \langle \Delta, s, T, D, RSS \rangle$. Constraints **LP1-LP4** of problem $ILP(P)$ provide the feasibility of semi-schedule $S$, while constraints **LP5 -LP8** provide its correspondence to pre-schedule $P$.

**Problem $ILP(P)$**

Find **integral non-negative** values of variables $x(q), y(q)$, satisfying the following constraints:

**LP1:** $x(q) \leq y(q),\ q \in [s]$ provides a non-negativeness of the length of each stay (which, however, admits the existence of zero-length stays);

**LP2:** $x(q) = 0,\ q \in [m]$ reflects the condition (d) on a feasible sequence $T$;

**LP3:** $\sum_{t \in [s_{k,\nu}]} (y(\beta(k,\nu,t)) - x(\beta(k,\nu,t))) \geq n_\nu,\ k \in [m],\ \nu \in [g_{ns}]$; it runs that the total length of all stays of each machine $M_k$ ($k \in [m]$) at any **non-special node** $v_\nu$ ($\nu \in [g_{ns}]$) cannot be less than the number of jobs located at that node;

**LP4:** $x(\gamma(k,t+1)) - y(\gamma(k,t)) = \rho(\nu(\gamma(k,t)), \nu(\gamma(k,t+1))),\ k \in [m],\ t \in [s_k - 1]$; the time needed to a machine $M_k$ for traveling between two its successive stays (that should be performed at **different nodes** of network $G^*$, by property (a) of component $T$ of a feasible pre-schedule) is defined for a given network by function $\rho$;

**LP5:** The following inequalities (that should hold for every $q \in [s-1]$) provide necessary and sufficient conditions for a schedule $S$ to meet the equality $T_{[S]} = T$ between the sequence of semi-stays $T_{[S]}$ (defined in accordance with (2)) and the

component $T$ of pre-schedule $P$.

$$x(q+1)-x(q)\geq\begin{cases}0, & \text{if } \mu(q)<\mu(q+1);\\ 1, & \text{otherwise.}\end{cases}$$

Indeed, the inequality $x(q+1)-x(q)\geq 0$ is necessary in all cases for the validity of the inequality $(b(q),\mu(q))<_{\text{lex}}(b(q+1),\mu(q+1))$; when $\mu(q)<\mu(q+1)$, it is also sufficient. In case $\mu(q)>\mu(q+1)$, the strict inequality $x(q+1)-x(q)>0$ is necessary and sufficient, which is equivalent (due to the integrality of variables) to the inequality $x(q+1)-x(q)\geq 1$. Finally, in case $\mu(q)=\mu(q+1)$, the inequality $x(q+1)-x(q)\geq 1$ is also sufficient, while its necessity follows from the requirement of feasibility of schedule $S$. This requirement (with respect to two successive stays of a machine) is formulated in a stronger form in **LP4**;

**LP6:** $y(\bar{q}(t))-x(\bar{q}(t))=A(P,t),\ t\in[\bar{s}]$;
it runs that the length of the $t$th special stay is defined by the amount $A(P,t)$ computable for a given relative schedule of special jobs ($RSS$) while specifying the pre-schedule $P$;

**LP7:** for each $t\in[2,\bar{s}]_{\mathbb{Z}}$, the constraint for the difference in time between the starting times of two successive special stays is specified by function $D$ of pre-schedule $P$:

$$x(\bar{q}(t))-x(\bar{q}(t-1))\begin{cases}=D(t), & \text{if } D(t)<2m;\\ \geq D(t), & \text{if } D(t)=2m;\end{cases}$$

**LP8:** $y(\gamma(k,s_k))\leq L(\Delta),\ k\in[m]$; the last stay of each machine (and henceforth, the whole schedule) has to be completed by the time $L(\Delta)$ defined in (5).

Problem $ILP'$ differs from $ILP$ by that only, the inequality **LP3** is replaced by the equality **LP3′**: $\sum_{t\in[s_{k,\nu}]}(y(\beta(k,\nu,t))-x(\beta(k,\nu,t)))=n_\nu,\ k\in[m],\ \nu\in[g_{ns}]$,

We denote:
$Sol(P)$ and $Sol'(P)$ to be the sets of solutions $\langle X,Y\rangle\doteq\{x(q),y(q)\,|\,q\in[s]\}$ of problems $ILP(P)$ and $ILP'(P)$, respectively, where $s$ is the total number of stays of all machines in pre-schedule $P$; $Sol(\Delta)=\cup_{P\in\mathcal{P}(\Delta)}Sol(P)$; $Sol'(\Delta)=\cup_{P\in\mathcal{P}(\Delta)}Sol'(P)$;
let $\Psi(X,Y):Sol(\Delta)\to\mathbb{Z}^+$ be the function defined by the equality:

$$\Psi(X,Y)\doteq\sum_{(x(q),y(q))\in\langle X,Y\rangle}(x(q)+y(q)).$$

The connection between two problems set above is established in the following

**Lemma 3.5.** *Given $\Delta\in[m]$, the relation $Sol(\Delta)\neq\varnothing$ holds,* ***if and only if*** *$Sol'(\Delta)\neq\varnothing$.*

*Proof.* **"if"** is evident, since every solution of problem $ILP'(P)$ is also a solution of problem $ILP(P)$.
**"only if"** Let $Sol(\Delta)\neq\varnothing$, i.e., the definition area of function $\Psi$ is nonempty. Since function $\Psi$ takes integral nonnegative values on the solutions $\langle X,Y\rangle\in Sol(\Delta)$, it attains its minimum on some solution $\langle X^*,Y^*\rangle\in Sol(P^*),\ P^*=\langle\Delta^*,s^*,T^*,D^*,RSS^*\rangle\in\mathcal{P}(\Delta)$. Let us prove that $\langle X^*,Y^*\rangle$ is also a solution of problem $ILP'(P^*)$.

Suppose the contrary, i.e., at least one of the inequalities $LP3$ (for some machine $M_{k'}$ and a non-special node $v_{\nu'}$) holds as a strict inequality. In view of the integrality of the solution $\langle X^*,Y^*\rangle$, the discrepancy in that inequality (the difference between its left and right parts) is not less than 1. Let us diminish it by decreasing by 1 the

length of the last nonempty stay of machine $M_{k'}$ at node $v_{\nu'}$ (i.e., by decreasing $y^*(q')$ by 1, where $q'$ is the absolute index of the stay). If the stay length decreases thereby down to zero (i.e., the stay is no longer a "stay", by Definition 2.10), we remove it and decrease by 1 the total number of stays ($s^*$). Next, if that stay was intermediate, we make the path of the machine from the node of its previous stay to the node of its next stay "straighter" (and, possibly, shorter). If, as a result of the above reduction, both those stays appear to be at the same node, we glue them together (according to the definition of a stay), reducing further the parameter $s^*$. Furthermore, if $q'$ is not the last stay of machine $M_{k'}$ (i.e., has got a *machine index* $t' < s_{k',\nu'}$), then to correspond to the requirement $LP4$, we diminish (by at least 1) the time $x^*(\gamma(k', t'+1))$ of the beginning of the next, $(t'+1)$th stay of machine $M_{k'}$, which may change both the order of stays in sequence $T^*$, and the function $D^*$. Besides, if the stay $(t'+1)$ is special (i.e., is situated in a special node $v_{\nu''}$), than the decreasing of its starting time results in a different relative schedule ($RSS'$), as well as increases the length of that stay. Thus, the new solution $\langle X', Y' \rangle$ is a solution of a problem $ILP(P')$, defined by another pre-schedule ($P' = \langle \Delta, s, T', D', RSS' \rangle$). If we prove that $P' \in \mathcal{P}(\Delta)$, we will obtain a contradiction with the choice of the solution $\langle X^*, Y^* \rangle$, since $\Psi(X', Y') < \Psi(X^*, Y^*)$.

The only circumstance that could prevent the pre-schedule $P'$ to belong to $\mathcal{P}(\Delta)$ is a possible violation (at least, for one operation $O_j^{k'}$ performed at the $(t'+1)$th stay of machine $M_{k'}$) of the restriction $LC_2(O_j^{k'}) \leq n_{\nu''} + \Delta - 1$ set on page 53 (in the definition of a relative schedule of special jobs). As a result of such a violation, the length of the stay would exceed $n_{\nu''} + \Delta - 1$, the idle time of the machine at node $v_{\nu''}$ (as well as its total idle time) would exceed $\Delta - 1$, and the length of the route would exceed $L(\Delta)$. Yet, it is clear that the length of semi-schedule $S'$, defined by the solution $\langle X', Y' \rangle$, coincides with the length of semi-schedule $S^*$ from which $S'$ was obtained, and thus, it cannot exceed $L(\Delta)$. The contradiction obtained proves the inclusion $P' \in \mathcal{P}(\Delta)$, which contradicts the choice of the solution $\langle X^*, Y^* \rangle$. Henceforth, it is proved that $\langle X^*, Y^* \rangle$ must be a solution of a problem $ILP'(P^*)$ for some $P^* \in \mathcal{P}(\Delta)$, whence $Sol'(\Delta) \neq \varnothing$. Lemma 3.5 is proved. □

By means of an original technique, Frank and Tardos (1987) elaborated an $FPT$-algorithm for solving a general *Integer Linear Programming problem* (*ILP-problem*); the algorithm was parameterized by the number of variables and used polynomially bounded space.

**Theorem 3.3.** (Frank and Tardos [8]; see **(author?)** [15, p. 27]) *Given an ILP-problem with $h$ variables, it can be solved in $O(h^{2.5h+o(h)}) \cdot |I_{ILP}| \leq 2^{O(h \log h)} \cdot |I_{ILP}|$ time and with a space requirement polynomial in the encoding length ($|I_{ILP}|$) of its input.*

To estimate the running time of the algorithm for solving the $ILP'(P)$-problem, as applied to our case, it is sufficient to estimate the number of variables and the encoding length of the input of the $ILP'(P)$-problem. It can be easily seen that both the number of variables and the number of constraints in problem $ILP'(P)$ are bounded above by the amount $O(s)$ which, due to the restrictions imposed on pre-schedules, cannot exceed $O(m^2 + mg)$. Since the matrix of coefficients of the $ILP'(P)$-problem consists of the numbers $\pm 1$ and 0 only, the record length of this matrix (in bits) coincides with the matrix size (i.e., $O(s^2)$); at that, the record of the

free column requires at most $O(s)\cdot|I|$ bits. Thus, we have $|I_{ILP}| \le O(s^2)+O(s)\cdot|I| \le O(s^2)\cdot|I|$, and due to the result of Theorem 3.3, we may formulate the following

**Lemma 3.6.** *Given a pre-schedule $P$, problem $ILP'(P)$ can be solved in time $2^{O(s\log s)}\cdot|I|$ and with a space requirement $Pol(m,g)|I|$, where $Pol(m,g)$ is a polynomial of $m$ and $g$; $s$ is the total number of stays in the solution (specified in $P$), and $|I|$ is the input length of the $ROS^*$-UET problem in the compact encoding scheme.*

### 3.5. The criterion of non-simultaneity of special operations and efficient algorithms of constructing a local job schedule at a non-special node.

**Proposition 3.4.** *If schedule $S$ is an extension of a pre-schedule $P = \langle\Delta, s, T, D, RSS\rangle \in \mathcal{P}$, then the processing intervals of any two special stays from distinct segments do not overlap.*

*Proof.* The validity of the statement directly follows from two facts:
(1) the total idle time of any machine in schedule $S$ is no greater than $m-1$ (since pre-schedule $P \in \mathcal{P}$ specifies the constraint: $C(S) \le L(\Delta) \doteq \bar{C}+\Delta-1 \le \bar{C}+m-1$); this implies that the length of any special stay in schedule $S$ does not exceed $2m-2$;
(2) the difference between the starting times of any stay of a segment and of any stay of the next segment is not less than $2m$ (by the definition of function $D$). □

**Lemma 3.7. (the criterion of non-simultaneity of special operations)** *Suppose, we are given a pre-schedule $P = \langle\Delta, s, T, D, RSS\rangle \in \mathcal{P}$. Then in any complete schedule $S$, being an extension of $P$, any two special operations $O'$ and $O''$ are **non-simultaneous** (by Definition 2.12), if and only if their segment coordinates differ in at least one component.*

*Proof.* **"only if"** Suppose that both segment coordinates of operations $O'$ and $O''$ coincide. This means that the completion times of these two operations are strictly tied to the starting time **of the same** special stay (with a relative number $SC_1(O') = SC_1(O'')$), and that they are shifted with respect to that moment by the same amount ($SC_2(O') = SC_2(O'')$),which implies the simultaneity of these operations.

**"If"** segment coordinates of operations $O'$ and $O''$ differ at their first components (i.e., the operations belong to different segments), then the time intervals of these operations do not overlap, by Statement 3.4. Suppose now that their first segment coordinates coincide, i.e., the operations $O'$ and $O''$ belong to the same segment, but differ in their second segment coordinates — in the shifts of their completion times with respect to the starting time of the segment. Then, due to the integrality of the relative schedule ($RSS$), the completion times of the operations $O'$ and $O''$ differ by at least 1. This provides their non-simultaneity, since both operations have unit length. □

Next, we will consider the problem of constructing a feasible local schedule for jobs located at a given non-special node within given time domains specified for each machine. In two independent lemmas, we will establish certain sufficient conditions that guarantee the existence of feasible local schedules. For a constructive proof of those lemmas, two procedures will be used. One of them is based on a theorem due to Cole et al (2001) and on their algorithm of finding a proper edge coloring of a bipartite graph in the minimum number of colors. Another procedure realizes

our algorithm of constructing a feasible schedule consistent with a given family of time domains. Although the second procedure has a more restricted scope of its implementation (it guarantees the construction of a desired schedule for large enough values of $n_\nu$ only), but exactly this procedure provides the $FPT$-property of the whole algorithm and the polynomial dependence of its running time on the length of recoding the problem input under the compact encoding scheme. As a result, both these procedures find their proper application in our algorithm.

**Definition 3.21.** We say that a local schedule of jobs at a non-special node $v_\nu$ *is consistent with a given family* $\mathcal{T}_\nu = \{\mathcal{T}_{k,\nu} \,|\, k \in [m]\}$ *of discrete domains of time* (see Definition 2.2), if: (a) the performance interval of each operation $O_j^k$ ($k \in [m],\ j \in \mathcal{J}(\nu)$) is contained in the domain $\mathcal{T}_{k,\nu}$; (b) any two *related operations* of jobs $\{j \in \mathcal{J}(\nu)\}$ are *non-simultaneous* (see Definitions 2.7 and 2.12).

**Definition 3.22.** We say that a schedule of jobs at a given node $v_{\nu'}$ ($\nu' \in [g]$) is *consistent with* a given feasible *solution* $\{x(q), y(q) \,|\, q \in [s]\}$ of the $ILP'(P)$-problem (defined for a given pre-schedule $P$), if it is consistent with the family $\mathcal{T}_{\nu'} = \{\mathcal{T}_{k,\nu'} \,|\, k \in [m]\}$ of discrete domains of time $\mathcal{T}_{k,\nu'} = \bigcup_{q \in Q(k,\nu')} [x(q), y(q)]$, where $Q(k, \nu') = \{q \in [s] \,|\, \mu(q) = k,\ \nu(q) = \nu'\}$.

**Theorem 3.4.** (Cole et al, 2001 [6]) *For any given bipartite multigraph with $h$ edges and with the maximum vertex degree $d$, there exists a proper edge coloring in $d$ colors. The coloring can be found in $O(h \log d)$ time.*

**Lemma 3.8.** *Let $\langle X, Y\rangle = \{x(q), y(q) \,|\, q \in [s]\}$ be a feasible solution of problem $ILP'(P)$, defined for a given pre-schedule $P \in \mathcal{P}$. Then for any non-special node $v_\nu$ there exists a local schedule $S(\nu)$ of jobs from $\mathcal{J}(\nu)$, consistent with the solution $\langle X, Y\rangle$ (see Definition 3.22). Schedule $S(\nu)$ can be constructed by Procedure 1 (described below) in time $O(mn_\nu \log n_\nu)$.*

*Proof.* Let $\mathcal{T}_{k,\nu}$ ($k \in [m],\ \nu \in [n_{ns}]$) be a discrete domain of time defined for a machine $M_k$ and a non-special node $v_\nu$ by the solution $\langle X, Y\rangle$ according to Definition 3.22. We should schedule $n_\nu$ operations of machine $M_k$ in the domain $\mathcal{T}_{k,\nu}$ satisfying the non-simultaneity conditions for all pairs of related operations. Since solution $\langle X, Y\rangle$ meets requirements $LP3'$, and the time intervals of all stays of machine $M_k$ do not overlap, domain $\mathcal{T}_{k,\nu}$ represents a union $n_\nu = \|\mathcal{T}_{k,\nu}\|$ of **different integral unit time intervals** $u_i = [i-1, i]$. Thus, to each machine $M_k$, it is assigned that many time units, that it needs for performing all jobs located at node $v_\nu$. In other words, the whole time assigned to machine $M_k$ for its stays at node $v_\nu$ should be spent to performing jobs; at that, in different units of time machine $M_k$ should perform different jobs; symmetrically, all jobs being performed at the same time unit must be different. As a result, we come to the following problem.

Let $B = (V_B, E_B),\ V_B = V_L \cup V_R$, be a bipartite graph in which the vertices of the left part ($v'_k \in V_L$) represent machines ($M_k,\ k \in [m]$), while the vertices of the right part ($v''_i \in V_R$) represent unit time intervals $\{u_i \,|\, i \in U(\nu) \doteq \cup_{k \in [m]} U_{k,\nu}\}$, $U_{k,\nu}$ is the set of indices of the unit time intervals that compound the domain $\mathcal{T}_{k,\nu}$. Let us connect each vertex $v'_k \in V_L$ by an edge with each vertex from $V_R$ representing an interval from $\{u_i \,|\, i \in U_{k,\nu}\}$. Thus, the cardinality of the left part is equal to $m$, while the degree of each vertex from the left part is equal to $n_\nu \geq m$; at that, the degree of each vertex from the right part is not greater than $m$, the total number

of edges is $mn_\nu$, and the maximum vertex degree in graph $B$ is equal to $n_\nu$. By Theorem 3.4, there exists a proper edge coloring of graph $B$ in $n_\nu$ colors, which can be found in $O(mn_\nu \log n_\nu)$ time. (This procedure of the proper edge coloring will be further referred to as "Procedure 1".) As one can guess, in this coloring, color $c(v'_k, v''_i)$ of an edge $(v'_k, v''_i)$ corresponds to the local index of the job being processed by machine $M_k$ in the unit time interval $u_i$. The absolute job index can be computed by the formula $j = N_{\nu-1} + c(v'_k, v''_i)$. □

It should be noted that, although the feasible local schedule of processing the jobs located at a given node $v_\nu$ is built by Procedure 1 in time $O(mn_\nu \log n_\nu)$ (which is polynomial on the number of objects at that node), this **is not a polynomial on the input length** in its compact encoding, because such encoding has length only logarithmically dependent on $n_\nu$. As a result, we may use the described above procedure only at nodes heaving **small values of the parameter** $n_\nu$ (for instance, values bounded by a function of parameters $m$ and $g$). For arbitrarily large values of $n_\nu$, in our algorithm quite a different Procedure 2 (described below) is used, capable to build rather simple feasible schedules, the encoding of which has length bounded by a polynomial of $\log n_\nu$. (And that much time will be spent in the algorithm for constructing those schedules. At that, the schedule remains uniquely and correctly defined.) Thus, Procedure 2 has the required running time. Everything seams to be Ok, but... the flaw of Procedure 2 is that it can guarantee the constructing of a proper schedule **only for large enough values** of $n_\nu$. The collision appeared is solved in the current paper as follows. All non-special nodes are divided into two categories: of *semi-special* nodes (with a number of objects $n_\nu < 2s(\nu)$, where $s(\nu)$ is the total number of stays of machines at node $v_\nu$) and — of *absolutely-non-special* nodes (with a number of objects $n_\nu \geq 2s(\nu)$), after which we apply Procedure 1 to the first nodes and Procedure 2 — to the second ones.

To describe and justify Procedure 2, we need to introduce a few notions.

**Definition 3.23** (COS)**.** The last operation of a stay (in a given schedule or in a schedule under construction) will be called a *completing operation of the stay* (or *COS*, for short).

As follows from properties (b) and (c) of schedules (from Lemma 3.1) and — from Corollary 3.1, any last operation of a stay completes that stay (in the sense that the completion time of the last operation coincides with the completion time of its personal stay).

**Definition 3.24** (conflicting operations)**.** We will say that two operations *are conflicting by job* in schedule $S$, if they are processed simultaneously (in the same unit time interval) and belong to the same job. Schedules of machines $M_k$ and $M_{k'}$ *are conflicting*, if there are two operations $O_j^k$ and $O_j^{k'}$ conflicting by job.

**Definition 3.25.** For a given $w \in \mathbb{N}$ we define two functions. Function $\varphi_w$ : $[w] \times [0, w-1]_{\mathbb{Z}} \to [w]$ specified by the formula

$$\varphi_w(i,h) \doteq \begin{cases} i-h, & \text{if } i > h \\ w+i-h, & \text{if } i \leq h \end{cases}$$

enables one, given an arbitrary permutation $\pi' = (\pi'(1), \ldots, \pi'(w))$ of indices $\{1, \ldots, w\}$, a given shift amount $h \in [0, w-1]_{\mathbb{Z}}$, and a position index $i \in [w]$, to compute the value $\pi''(i) = \pi'(\varphi_w(i,h))$ of the permutation $\pi''$ obtained from $\pi'$ via

a cyclic shift by amount $h$ (see Definition 2.3). Function $\eta_w : [w] \times [w] \to [0, w-1]_{\mathbb{Z}}$ specified by formula

$$\eta_w(i,j) \doteq \begin{cases} i-j, & \text{if } i \geq j \\ w+i-j, & \text{if } i < j \end{cases}$$

defines the shift $h$ of permutation $\pi''$ with respect to $\pi'$ providing the equality $\pi''(i) = \pi'(j)$. (Thus, $\forall\, i,j \in [w]$, we have the identity: $j = \varphi_w(i, \eta_w(i,j))$.)

**Lemma 3.9.** *Let $\langle X, Y\rangle = \{x(q), y(q) \,|\, q \in [s]\}$ be a feasible solution of the $ILP'(P)$-problem defined for a given pre-schedule $P \in \mathcal{P}$. Then for any ANS-node $v_\nu$ (see Definition 2.16, page 52) there exists a local schedule $S(\nu)$ for the jobs from $\mathcal{J}(\nu)$ consistent with the solution $\langle X, Y\rangle$ (see Definition 3.22). Schedule $S(\nu)$ can be constructed by means of Procedure 2 (described below) with a bit running time $O(ms(\nu) + s(\nu)\log s(\nu)) \cdot |I|$, where $|I|$ is the input length of the given instance $I$ of problem ROS*-UET in the compact encoding scheme.*

*Proof.* Let $\mathcal{T}_{k,\nu}$ ($k \in [m]$, $\nu \in [n_{ns}]$) be a discrete domain defined for machine $M_k$ and for an ANS-node $v_\nu$ by a solution $\langle X, Y\rangle$ according to Definition 3.22. Since solution $\langle X, Y\rangle$ meets requirements $LP3'$, and once the intervals $\{[x(q), y(q)]\}$ constituting the domain $\mathcal{T}_{k,\nu}$ do not overlap, we have the equality $\|\mathcal{T}_{k,\nu}\| = n_\nu$.

Given an ANS-node $v_\nu$, let us renumber the machines $\{M_k\}$ in the non-increasing order of $s_{k,\nu}$, while for jobs from $\mathcal{J}(\nu)$ we will use their original (*local*) numeration by indices from $[n_\nu]$. Thus, the processing of the jobs from $\mathcal{J}(\nu)$ by a machine $M_k$ can be specified by a permutation $\pi_k$ of indices $\{1, 2, \ldots, n_\nu\}$; in view of the equality $\|\mathcal{T}_{k,\nu}\| = n_\nu$, each such permutation uniquely defines a schedule $S_k$ of performing the jobs within the domain $\mathcal{T}_{k,\nu}$, since the machine has no idle time within that domain. The main peculiarity of schedule $S(\nu)$ (to be constructed) is that each permutation $\pi_k$ ($k \in [m]$) will be obtained from the permutation $\pi_1 \doteq (1, 2, \ldots, n_\nu)$ via a cyclic shift by a certain amount $h_k$ (see Definition 2.3).

The amounts $\{h_k\}$ will be defined by induction on $k \in [m]$ in the order of machine numeration. We set $h_1 = 0$. Suppose that for some $k > 1$ the amounts $h_1, \ldots, h_{k-1}$ are already defined so that schedules $S_1, \ldots, S_{k-1}$ meet the non-simultaneity requirements (and thus, there are no conflicts between those schedules). Let us show that there exists a proper value of the amount $h_k$, which defines a schedule $S_k$ conflicting with no of already defined schedules $S_1, \ldots, S_{k-1}$.

Suppose that the amount of the shift $h_k$ is somehow defined, thereby defining a permutation $\pi_k$, and suppose that in the partial schedule specified by permutations $\pi_1, \ldots, \pi_k$, a conflict appeared between two operations ($O^k_{j'}$ and $O^{k'}_{j'}$, $k' < k$) of a job $j' \in \mathcal{J}(\nu)$ which was scheduled to be processed in stays $T(q)$ and $T(q')$ of machines $M_k$ and $M_{k'}$ simultaneously. Since in the schedules of both machines the jobs from $\mathcal{J}(\nu)$ are processed in the same order $(1, 2, \ldots, n_\nu)$ (but with, maybe, different cyclic shifts), this implies that the succeeding operations on these machines (within the same pair of stays) are also pairwise conflicting. As a result, we obtain one of two possible situations: (a) the COS defined (see Definition 3.23) for stay $T(q')$ is conflicting with some operation of stay $T(q)$; (b) the COS defined for stay $T(q)$ is conflicting with some operation of the same job which is simultaneously processed in stay $T(q')$. Thus, to choose a feasible value of the amount $h_k$, it is sufficient to avoid the conflicts of the types (a) and (b).

While analyzing the conflict situations of type (a), we observe that each COS of machines $M_1, \ldots, M_{k-1}$ hits into the interval of at most one stay of machine $M_k$,

and if such a hitting happens for some COS $O_j^{k'}$, then its conflict with operation $O_j^k$ appears in that case only, when the latter is being performed in the same unit time interval. At that, the hitting of $O_j^k$ into that unit interval happens for a unique value of the shift $h_k$. Thus, each COS of machines $M_1, \ldots, M_{k-1}$ (at node $v_\nu$ under consideration) defines at most one restriction on the choice of the proper value of $h_k$ (at that, different COSes may define the same restriction). We may conclude that to avoid the conflict situations of type (a), it is sufficient to forbid at most $s(\nu) - s_{k,\nu}$ variants of values $h_k$.

While analyzing in a similar way the conflict situations of type (b), we observe that each COS of machine $M_k$ conflicts with at most $k-1$ operations of machines $M_1, \ldots, M_{k-1}$. Thus, the conflict situations of type (b) define at most $(k-1)s_{k,\nu}$ restrictions on the choice of the proper value of $h_k$, which together with conflicts of type (a) results in at most $z_{k,\nu} \doteq s(\nu) + (k-2)s_{k,\nu}$ restrictions. Since the machines are indexed by non-increasing of $s_{k,\nu}$, for any $k$ the inequality $s_{k,\nu} \le s(\nu)/k$ holds, whence we obtain the bound $z_{k,\nu} \le s(\nu) + \frac{k-2}{k}s(\nu) < 2s(\nu)$. Since we have got $n_\nu$ possible values of $h_k$, and since for any ANS-node (by Definition 2.16) we have the inequality $n_\nu \ge 2s(\nu)$, it follows that for each machine $M_k$ there always exists a **non-conflict** variant of the shift $h_k$.

To complete the proof of the lemma, it remains to describe the algorithm (called a *Procedure 2*) of finding a feasible schedule with the claimed bound on running time. The algorithm consists in searching for a proper (non-conflict) value of the shift $h_k$ for each machine $M_k$, $k \in [m]$, and in the subsequent presentation of the schedule in a compact form.

### Procedure 2

At the **input** of the procedure, we are given a family $B_\nu = \{B_{k,\nu} \,|\, k \in [m]\}$ of finite sets $B_{k,\nu} \doteq \{(x(q), y(q)) \,|\, q \in Q(k,\nu)\}$, specifying the limits of stays of each machine $M_k$ ($k \in [m]$) at node $v_\nu$, where $Q(k,\nu) = \{\beta(k,\nu;t) \,|\, t \in [s_{k,\nu}]\}$ is the set of absolute indices of those stays. Each set $B_{k,\nu}$ specifies a discrete domain $\mathcal{T}_{k,\nu} = \cup_{q \in Q(k,\nu)}[x(q), y(q)]$.

At the **output** we have a schedule for each machine $M_k$ at a given ANS-node $v_\nu$ in the form of a list of pairs $S_{k,\nu} = \{\langle \tau_{k,\nu}(t), \mathcal{J}_{k,\nu}(t)\rangle \,|t \in [s'_{k,\nu}]\}$, where $\tau_{k,\nu}(t) = [\tau'_{k,\nu}(t), \tau''_{k,\nu}(t)]$ is a time interval in which machine $M_k$ performs (without an idle time) all jobs with indices from the interval $\mathcal{J}_{k,\nu}(t) = [j'_{k,\nu}(t), j''_{k,\nu}(t)]_{\mathbb{Z}}$ in the increasing order of indices, and the value of the parameter $s'_{k,\nu}$ is defined in the procedure and is equal to one of two values: $\{s_{k,\nu}, s_{k,\nu}+1\}$.

Beside that, the following information is used in the "body" of the procedure:
— the list $\mathcal{E}$ of *events* listed in a non-increasing order of their occurrence times (backwards). As "events", we consider the start or the completion of a stay at node $v_\nu$. Each event $e \in \mathcal{E}$ is characterized by five parameters: $\langle k(e), t(e), q(e), st(e), \tau(e)\rangle$, where $k(e)$ is the machine number, $t(e)$ and $q(e)$ are the local and the absolute indices of the stay, $st(e)$ is the "status" of the event ($st(e)$ = "start" or "completion" of the stay);

$$\tau(e) = \begin{cases} y(q(e)), & \text{if } st(e) = \text{"completion"} \\ x(q(e)), & \text{if } st(e) = \text{"start"} \end{cases} \quad \text{— is the occurrence time of event } e;$$

— array $H[1..m]$, where $H[k]$ specifies the amount $h_k$ of the cyclic shift of permutation $\pi_k$ with respect to permutation $\pi_1$;

— a bit array $F[1..2s(\nu)]$ of forbidden values of $h_k$ at step $k$ of the *Main stage*, where $F[i] = 1$, if the value $h_k = i - 1$ is forbidden, and $F[i] = 0$ — otherwise; (since the maximum possible number of forbidden values is less than $2s(\nu)$, it is sufficient to check the values $h_k \in [0, 2s(\nu) - 1]_{\mathbb{Z}}$ only; among them, there exists (for sure) at least one admissible value);
— array $SM[1..m]$ (the "status" of machine $M_k,\ k \in [m]$); $SM[k]$ takes values from the interval $[0, s_{k,\nu}]_{\mathbb{Z}}$; $SM[k] = 0$ means that at the time of the occurrence of the current event $e \in \mathcal{E}$ machine $M_k$ has no stay at node $v_\nu$; a value $SM[k] = t > 0$ means that the machine performs its $t$th stay at node $v_\nu$ (where $t$ is the *local index of the stay*);
— array $N'[1..m, 1..(g + m/2)]$ keeps positions of COSes in permutations $\{\pi_k\}$ over all $k \in [m]$ and over all stays of machine $M_k$ at node $v_\nu$. The number of such stays $(s_{k,\nu})$ is not greater than $g + m/2$, in view of property (c) of the sequence of stays $T$ (specified in the pre-schedule) and in view of the restriction $\Delta \le m$.

The procedure consists of two stages: the *Preliminary* and the *Main* one.

At the **Preliminary stage**, we make the initialization of values of the arrays that are to be computed. Arrays $F, SM$, and $N'$ are set identically to zeros; $\mathcal{E} :=$ **null**. Furthermore, for all $k \in [m]$ and $t \in [s_{k,\nu}]$, we compute:

$$N'[k,t] := \sum_{q \in Q(k,\nu,t)} (y(q) - x(q)),\ Q(k,\nu,t) \doteq \{\beta(k,\nu;u) \,|\, u \in [t]\}.$$

**The Main stage** represents a cycle on $\tilde{k} = 1, ..., m$, where at step $\tilde{k}$ we compute the values of $H[\tilde{k}]$ and construct the schedule of machine $M_{\tilde{k}}$ at node $v_\nu$.

**Step $\tilde{\mathbf{k}}$** consists of four *actions*, denoted by letters "***A**i*". (At step $\tilde{k} = 1$ action ***A2*** is skipped, as a result of which the array $F$ of forbidden values remains equal to zero, and for the shift of permutation $\pi_1$ the value $H[1] = 0$ is chosen.)

***A1*** (the adjustment of the *list of events* $\mathcal{E}$: adding to $\mathcal{E}$ the events of machine $M_{\tilde{k}}$, with a simultaneous renumbering of the list by lexicographical increasing of pairs $(-\tau(e), k(e))$). Look through the list $B_{\tilde{k},\nu} = \{(x(q'), y(q')) \,|\, q' = \beta(\tilde{k}, \nu; t'),\ t' \in s_{\tilde{k},\nu}\}$ of the time limits of stays of machine $M_{\tilde{k}}$; for each $q'$, we define two events: $\{e'_{q'}, e''_{q'}\}$ (of the start and of the completion of stay $q'$). Put $k(e'_{q'}) = k(e''_{q'}) := \tilde{k}$; $t(e'_{q'}) = t(e''_{q'}) := t'$; $q(e'_{q'}) = q(e''_{q'}) := q'$; $st(e'_{q'}) :=$ **"start"**; $st(e''_{q'}) :=$ **"completion"**; $\tau(e'_{q'}) := x(q') + \frac{1}{2}$; $\tau(e''_{q'}) := y(q')$.

Insert each new event $e$ into the list $\mathcal{E}$, while keeping the increasing order of pairs $(-\tau(e), k(e))$.

***A2*** (formation of the array $F[1..2s(\nu)]$ of the forbidden values for $H[\tilde{k}]$). We are consecutively (in the cycle on $\ell := 1, \ldots, |\mathcal{E}|$) looking through the list of events $e \in \mathcal{E}$, while establishing the value ($\tilde{\tau}$) of the *current time* (running conversely): $\tilde{\tau} := \tau(e)$.

If $st(e) =$"**completion**", we put $SM[k(e)] := t(e)$ (which means: "the stay $t(e)$ of machine $M_{k(e)}$ is *open*"). In that case, several forbidden values of $H[\tilde{k}]$ can be found, preventing possible conflicts of the COS defined for stay $t(e)$ of machine $M_{k(e)}$ with operations of the same job on other machines. Two possible cases are considered: $k(e) \ne \tilde{k}$ and $k(e) = \tilde{k}$.

If $k(e) \ne \tilde{k}$, then for arising a possible conflict (of type (a)) with machine $M_{\tilde{k}}$, the condition $t' \doteq SM[\tilde{k}] > 0$ must hold (which means: at the current point in

time ($\tilde{\tau}$) a stay $t'$ of machine $M_{\tilde{k}}$ is open). For defining a forbidden value, we need to know the index of the job ($j$) being processed on machine $M_{k(e)}$ in the unit time interval $u_{\tilde{\tau}}$. By Definition 3.25, the index of the job that stays in the $i$th position of permutation $\pi''$ (shifted by $h$ with respect to $\pi'$), coincides with the index of the job staying in position $\varphi_w(i,h)$ of permutation $\pi'$. Since in our case, $w = n_\nu,\ h = H[k(e)],\ i = N'[k(e),t(e)],\ \pi'' = \pi_{k(e)},\ \pi' = \pi_1 = (1,2,\ldots,n_\nu)$ (whence $\pi'(i') = i'$), we obtain

$$j = \pi''(i) := \varphi_{n_\nu}(N'[k(e),t(e)], H[k(e)]).$$

For a conflict on job $j$ to be arisen between machines $M_{k(e)}$ and $M_{\tilde{k}}$, the same job $j$ must be processed on machine $M_{\tilde{k}}$ in the time interval $u_{\tilde{\tau}}$. Since this unit time interval corresponds to the position $i' \doteq N'[\tilde{k},t'] - y(\beta(\tilde{k},\nu;t')) + \tilde{\tau}$ in permutation $\pi_{\tilde{k}}$, it remains to determine the shift of permutation $\pi_{\tilde{k}}$ at which job $j$ jumps to position $i'$. The shift can be computed by the formula $h := \eta_{n_\nu}(i',j)$ (see Definition 3.25 of function $\eta_w$). To make this value of the shift forbidden (in case $h < 2s(\nu)$), we put $F[h+1] := 1$.

In the alternative case, when $k(e) = \tilde{k}$, several (up to $\tilde{k}-1$) new conflicts of type (b) may arise between the machine $M_{\tilde{k}}$ and some machines with smaller indices. For discovering possible conflicts, we look through (in a cycle on $k < \tilde{k}$) the elements of the array $SM[k]$. If for some $k$ the inequality $SM[k] = t' > 0$ holds, this means that there may be a conflict between machines $M_k$ and $M_{\tilde{k}}$. To compute the forbidden value of the shift $H[\tilde{k}]$, we first determine the number of the position ($i'$) of permutation $\pi_k$, corresponding to the unit time interval $u_{\tilde{\tau}}$. It can be computed by the already known formula: $i' := N'[k,t'] - y(\beta(k,\nu;t')) + \tilde{\tau}$. Thus, we learn that in the time interval $u_{\tilde{\tau}}$ on machine $M_k$ job $j = \varphi_{n_\nu}(i', H[k])$ is being performed. Now the value of the shift ($h$), at which job $j$ falls into the position $i'' \doteq N'[\tilde{k},t(e)]$ of permutation $\pi_{\tilde{k}}$, can be computed by the formula $h := \eta_{n_\nu}(i'',j)$; in case $h < 2s(\nu)$ this enables us to set a new forbidden value of the shift of permutation $\pi_{\tilde{k}}$: $F[h+1] := 1$.

If $st(e) =$"**start**", we put $SM[k(e)] := 0$ (which means: "stay $t(e)$ of machine $M_{k(e)}$ is *closed*").

By the completion of scanning the list of the events ($\mathcal{E}$), array $F[1..2s(\nu)]$ contains all possible forbidden values of the shift of permutation $\pi_{\tilde{k}}$ from the interval $[0, 2s(\nu)-1]$.

***A3*** (the choice of the feasible value of the shift $H[\tilde{k}]$). By looking the array $F[1..2s(\nu)]$ through, we find $i' = \min\{i \in [2s(\nu)] \mid F[i] = 0\}$ and put $H[\tilde{k}] := i' - 1$. (As was shown above, such a value $i'$ exists for sure.)

***A4*** (constructing a feasible schedule for machine $M_{\tilde{k}}$ at node $v_\nu$, not conflicting with the schedules of machines $M_1,\ldots,M_{\tilde{k}-1}$). As shown above, the value of the shift $H[\tilde{k}]$ uniquely defines both the permutation $\pi_{\tilde{k}}$, and the schedule of machine $M_{\tilde{k}}$ at node $v_\nu$. To find that schedule, we first compute the position $i'$ of the job with index $n_\nu$ in permutation $\pi_{\tilde{k}}$:

$$i' = \begin{cases} H[\tilde{k}], & \text{if } H[\tilde{k}] > 0 \\ n_\nu, & \text{if } H[\tilde{k}] = 0. \end{cases}$$

Find the local index ($t' \in [s_{\tilde{k},\nu}]$) of the stay in which machine $M_{\tilde{k}}$ performs this job: $t' = \min\{t \in [s_{\tilde{k},\nu}] \mid N'[\tilde{k},t] \geq i'\}$. (Such $t$ exist: e.g., $N'[\tilde{k}, s_{\tilde{k},\nu}] = n_\nu \geq i'$.) Put

$d := N'[\tilde{k}, t'] - i'$. Next, for each stay $t \neq t'$ of machine $M_{\tilde{k}}$, we form the interval of indices $[j'_{\tilde{k},\nu}(t), j''_{\tilde{k},\nu}(t)]_{\mathbb{Z}} \doteq \mathcal{J}_{\tilde{k},\nu}(t)$ of those jobs that are to be processed in that stay in the time interval $\tau_{\tilde{k},\nu}(t) \doteq [\tau'_{\tilde{k},\nu}(t), \tau''_{\tilde{k},\nu}(t)] = [x(q(t)), y(q(t))],\ q(t) = \beta(\tilde{k}, \nu; t)$. We put

$$\begin{aligned} d_t &\doteq y(q(t)) - x(q(t)); \\ j''_{\tilde{k},\nu}(t) &:= \varphi_{n_\nu}(N'[\tilde{k}, t], H[\tilde{k}]); \\ j'_{\tilde{k},\nu}(t) &:= j''_{\tilde{k},\nu}(t) - d_t + 1. \end{aligned}$$

For the stay with index $t = t'$, we define: $\mathcal{J}_{\tilde{k},\nu}(t') = [n_\nu - d_{t'} + d + 1, n_\nu]_{\mathbb{Z}}$ to be the interval of job indices being processed in the time interval $\tau_{\tilde{k},\nu}(t') \doteq [\tau'_{\tilde{k},\nu}(t'), \tau''_{\tilde{k},\nu}(t')] = [x(q(t')), y(q(t')) - d]$. If at that, $d = 0$, then we are done. Otherwise (when $d > 0$), we define one more, $(s_{\tilde{k},\nu} + 1)$th interval of job indices: $\mathcal{J}_{\tilde{k},\nu}(s_{\tilde{k},\nu} + 1) = [d]$, of jobs being processed in the time interval $\tau_{\tilde{k},\nu}(s_{\tilde{k},\nu} + 1) = [y(q(t')) - d, y(q(t'))]$.

In the schedule of machine $M_{\tilde{k}}$ defined above, we used "local" job indices at node $v_\nu$. To complete the construction of the schedule for machine $M_{\tilde{k}}$, it remains to pass over to the *natural numeration of jobs* (see Definition 2.9), just by increasing the defined above limits on job indices by the amount $N_{\nu-1}$. ■

Next, we should prove that the schedule constructed is feasible. In our opinion, the only place in the algorithm that needs additional explanation is a possible "lost" of some forbidden values while performing the Action 2 in the case, when two events with status "completion" happen simultaneously. We mean the following situation.

Suppose that at the completion of Action 1 at step $k''$ of the cycle on $\tilde{k}$, two simultaneous events ($e'$ and $e''$) with indices $\ell'$ and $\ell''$ (in the list $\mathcal{E}$) and with the status of the "completion" of stays $q(e')$ and $q(e'')$, respectively, appear in the list of events; at that, we have $k(e') < k(e'') = k''$, whence $\ell' < \ell''$. Then, while scanning the event $e'$ (at Action 2), the interval of stay $q(e'')$ is not "opened" yet, and so, at step $\ell'$ of the cycle on $\ell$, we do not find a possibility for a possible conflict between the COS defined for stay $q(e')$ and the COS defined for stay $q(e'')$, being performed in the same unit time interval. However, the possibility for such a conflict is analyzed while scanning the event $e''$ (when stay $q(e')$ is still "opened").

An opposite situation (when we could define a redundant forbidden value, and as a consequence, could get lost the existing solution) could arise in the case, when two events appear ($e'$ and $e''$) with opposite statuses ($st(e')$ ="completion", and $st(e'')$ ="start" of a stay), and when, additionally, $\ell' < \ell''$ in $\mathcal{E}$. Then while scanning the event $e'$, stay $t(e'')$ would be still "opened". Yet we avoid such situations by a small increasing (by amount $\delta = 1/2$) of the occurrence times of the events with the "start" status.

Thus, we have considered all situations with a possible "incorrectness" of the algorithm. It remains to estimate its running time.

Since we define two events for each of $s(\nu)$ stays, at each step of the algorithm we have $|\mathcal{E}| = O(s(\nu))$. Each event should be inserted into a proper place of the list $\mathcal{E}$, which can be done in $O(\log s(\nu))$ time. Thus, the total time consumed by Action 1 (over all steps of the cycle on $\tilde{k}$) cannot exceed $O(s(\nu) \log s(\nu))$.

While performing Action 2 at each step of the cycle on $\tilde{k}$, for each event $e \in \mathcal{E}$ with $k(e) \neq \tilde{k}$, a constant number of calculations has to be performed (or $O(ms(\nu))$ operations in total, over all steps of the cycle on $\tilde{k}$). For an event $e$ with $k(e) = \tilde{k}$, the checking of the values of $SM[k]$ for all $k < \tilde{k}$ and the subsequent forming of forbidden values requires $O(m)$ time, which provides in total (over all steps of the cycle on $\tilde{k}$) the same bound $O(ms(\nu))$. That many bound is valid on the total running time of A2 (over all steps of the cycle on $\tilde{k}$). The total running time of Action 3 cannot also exceed $O(ms(\nu))$. Finally, the total running time of Action 4 (over all steps of the cycle on $\tilde{k}$) is estimated by $O(s(\nu))$. The overall number of computational operations of Procedure 2 amounts $O(s(\nu)(\log s(\nu)+m))$. To estimate its running time (subject to the encoding length of the operands of computational operations), this amount should be multiplied by $|I|$. Lemma 3.9 is proved. □

### 3.6. **The criterion of existing a complete feasible schedule of a given length.**

**Lemma 3.10.** *Given $\Delta \in [m]$, there exists a complete feasible schedule $S$ of length $C(S) \leq L(\Delta)$, **if and only if** $Sol'(\Delta) \neq \varnothing$.*

*Proof.* **"if"** Suppose that for a given pre-schedule $P = \langle \Delta, s, T, D, RSS \rangle \in \mathcal{P}(\Delta)$ there exists a solution $\langle X, Y \rangle = \{x(q), y(q) \,|\, q \in [s]\}$ of problem $ILP'(P)$. This solution specifies the time intervals $[b(q), e(q)] = [x(q), y(q)]$ for all stays $\{T(q) \,|\, q \in [s]\}$ in pre-schedule $P$. Thus, the complete information on machine routes in network $G^*$ becomes known. Furthermore, by the given relative schedule of special jobs and by the known starting times $\{b(\bar{q}(t)) \,|\, t \in [\bar{s}]\}$ of special stays we can uniquely restore the absolute schedule for special jobs (by formula (6)). It can be easily shown that this schedule is consistent in each special node $v_{\nu'}$ with the solution $\langle X, Y \rangle$ (according to Definition 3.22).

Indeed, property (a) from Definition 3.21 is satisfied, since the relative (and therefore, the absolute) schedule is defined **for all** special operations from $\mathcal{O}(\nu')$. Property (b) (the inclusion of the processing interval of operation $O_j^k$ into one of the intervals $\{[b(q), e(q)] \,|\, q \in Q(k, \nu')\}$) is a corollary of the following properties:
— the position of the processing interval of any special operation with respect to the interval of its personal stay is uniquely defined by the relative schedule and does not depend on the subsequent specification of the position of special stays (in the solution of problem $ILP'(P)$);
— in the relative schedule, the completion time of an operation $O_j^k$ is shifted (to the right) with respect to the starting time of its personal stay (with a relative index $t \in [\bar{s}]$) by the amount $LC_2(O_j^k)$ taking integer nonnegative values not less than 1;
— the processing time of the operation is equal to 1, henceforth, the whole interval of processing the operation lays to the right of the stay starting time;
— the completion time of stay $t$, in view of the requirement $LP6$ of problem $ILP'(P)$, is shifted with respect to the stay starting time $t$ by the amount $A(P, t)$;
— the amount $A(P, t)$, by its definition (7), is not less than the second local coordinate $(LC_2(O_j^k))$ of any operation $O_j^k \in \mathcal{O}(\nu')$ being processed in stay $t$; thus, the whole processing interval of the operation lays to the left of the completion time of its personal stay.

And lastly, property (c) (the non-simultaneity of related operations from $\mathcal{O}(\nu')$) is being checked at once while forming a variant of the relative schedule. Thus, the

consistency of the schedule of special jobs with the solution of problem $ILP'(P)$ is proved.

Next, given a semi-schedule, to extend it to a complete schedule, it is required (and sufficient) to construct at each special node $v_\nu$ a local schedule of jobs from $\mathcal{J}(\nu)$, consistent with the solution of problem $ILP'(P)$. The existence of such a schedule at each non-special node $v_\nu$ was proved in Lemma 3.8.

To prove the feasibility of the **complete schedule** $S$ (just obtained), we need to make sure of the non-simultaneity of related operations. For pairs of operations belonging to the same set $\mathcal{O}(\nu)$, this was proved in Lemma 3.8. Alternatively, if two related operations are located at different nodes, then they are "related by machine", and are performed in **different stays** of that machine. In that case, the non-simultaneity of the operations is a consequence of the non-overlapping the stays of each machine (which follows from the requirements $LP1$ and $LP4$ of problem $ILP'(P)$).

Finally, $C(S) \leq L(\Delta)$, since $S$ meets $LP8$.

**"only if"** Suppose that for a given $\Delta \in [m]$ there exists a complete feasible schedule $S$ of length $C(S) \leq L(\Delta)$. Then the optimal schedule $S^*$ with properties $P1$-$P9$ (the existence of which is proved in Lemma 3.3) has length $C(S^*) = L(\Delta_{[S^*]}) \leq C(S) \leq L(\Delta)$, whence $\Delta_{[S^*]} \leq \Delta$. By Proposition 3.3, we have $P_{[S^*]} \in \mathcal{P}(\Delta_{[S^*]}) \subseteq \mathcal{P}(\Delta)$. Therefore, since schedule $S^*$ provides a feasible solution to problem $ILP(P_{[S^*]})$, we have $Sol(\Delta) \neq \varnothing$, and by Lemma 3.5, $Sol'(\Delta) \neq \varnothing$. □

## 4. A description and justification of the algorithm $\mathcal{A}$

4.1. **A basic idea and an informal scheme of the algorithm.** The algorithm for solving the $ROS^*$-$UET$ problem is based on Lemma 3.10, from which it follows that, to construct an optimal schedule, it is sufficient to perform the following steps.

— Enumerate all pre-schedules from a certain limited domain (the size of which is bounded above by a function of $m$ and $g$), and for each pre-schedule $P$, try to solve problem $ILP'(P)$ by the algorithm of (Frank and Tardos, 1987) with the running time bounded by a function of $m$ and $g$. (With that purpose, we preliminarily extract from $P$ all necessary information, needed to form constraints in problem $ILP'(P)$.) The enumeration of pre-schedules completes, as soon as the first pre-schedule $P$ is found for each there exists a feasible solution of the $ILP'(P)$-problem.

— The *semi-schedule* obtained from the solution of problem $ILP'(P)$ should be extended up to a feasible *complete schedule* $S$ via constructing at each node of network $G^*$ (for the jobs located at that node) a feasible local schedule consistent with the solution of problem $ILP'(P)$. Such schedules are to be built by means of three different ways for three types of nodes: *special* ones, *semi-special*, and *absolutely-non-special*.

— For the nodes of the first type, the *relative schedule* is used (specified in pre-schedule $P$) and the solution of problem $ILP'(P)$.

— For each node of the second type, we find a proper edge coloring of a bipartite graph by a minimum number of colors (the existence of such coloring by the number of colors equal to the number of jobs located at a given non-special node is guaranteed by Lemma 3.8); the coloring obtained is then transformed into a schedule of jobs.

— At each node of the third type, for constructing a feasible schedule consistent

with the solution of problem $ILP'(P)$, we apply Procedure 2 from the proof of Lemma 3.9.

Since pre-schedules are enumerated in the algorithm in a non-decreasing order of the parameter $\Delta$ (and so, by a non-decreasing of the upper bound $L(\Delta)$ on schedule length), it follows that the very first feasible schedule constructed will be optimal.

4.2. **A detailed scheme of the algorithm.** 1. Preliminary computation:
(a) Constructing the reduced transportation network $G^* = (V^*, E^*)$.
(b) Searching for the hamiltonian cycle $(H^*)$ of the minimum length $\rho(H^*)$ in network $G^*$.
(c) Computing the lower bound on the optimum $(\bar{C})$ of problem $ROS^*$-$UET$.

% The beginning of the **Big Cycle**. Assigning values to components of a pre-schedule.
2. **Begin of the cycle** on $\Delta := 1, \ldots, m$.
Computing the upper bound $L(\Delta) = \bar{C} + \Delta - 1$ on schedule length,
the upper bound $2g - 2 + \Delta$ on the number of stays of any machine,
and the upper bound $\hat{s} = m(2g - 2 + \Delta)$ on the total number of stays.
3. **Begin of the cycle** on $s := m(g+1), \ldots, \hat{s}$ % specifying the total number of stays.
4. **Begin of the cycle** on variants of sequence $T$ (the total sequence of semi-stays) of the given length $s$.
5. Computing a "supplementary information" for the given variant of $T$.
6. Verification of the variant of $T$ on its correctness. (The iteration of the cycle on $T$ is terminated ahead of time, if one of the necessary conditions is unsatisfied.)
7. **Begin of the cycle** on variants of function $D: [\bar{s}+1] \to \{0, 1, \ldots, 2m\}$.
8. Computing a "supplementary information" for the given variant of function $D$ (including the information on the *segmentation* of the sequence of special stays).
9. **Begin of the cycle** on variants of $RSS$ (the relative schedule of special jobs).
10. Computing a "supplementary information" (such as segment coordinates of special operations and lengthes of special stays).
11. Verification of the variant of $RSS$ on its correctness. The iteration of the cycle on $RSS$ is terminated ahead of time, if one of the necessary conditions is unsatisfied.
12. Generating the constraints of problem $ILP'(P)$ for the given variant of pre-schedule $P$. Solving the problem $ILP'(P)$.
13. If a **feasible solution** $\{x(q), y(q) \,|\, q \in [s]\}$ of problem $ILP'(P)$ is found, we specify the time limits of all stays $(b(q) := x(q);\ e(q) := y(q),\ q \in [s])$ and go out of the Big Cycle and go to the label "***B1:***" (to the procedures of constructing local schedules at nodes of network $G^*$, using the semi-schedule found in the $ILP'(P)$-problem).
14. **End** of the iteration of the cycle on variants of $RSS$
15. **End** of the iteration of the cycle on variants of $D$
16. **End** of the iteration of the cycle on variants of $T$
17. **End** of the iteration of the cycle on variants of $s$
18. **End** of the iteration of the cycle on variants of $\Delta$
% **End** of the iterations of the Big Cycle.
19. ***B1:*** % The *Final stage* (procedures of constructing the local schedules):
20. Constructing the local schedules at special nodes $v_\nu$ $(n_\nu < m)$.
21. Constructing the local schedules at semi-special nodes $v_\nu$ $(m \le n_\nu < 2s(\nu))$ by

means of Procedure 1.
22. Constructing the local schedules at absolutely-non-special nodes $v_\nu$ ($n_\nu \geq 2s(\nu)$) by means of Procedure 2.

4.3. **A description and justification of the algorithm procedures.**
***(line 1) Preliminary computation:***
(a) Constructing the reduced transportation network $G^* = (V^*, E^*)$:
scanning the set $V$ of nodes of the original network $G$; selecting a subset $V^* \subseteq V$ of active nodes (i.e., the depot and all nodes with $n_\nu > 0$);
numbering the nodes of the set $V^*$ in the non-increasing order of $n_\nu$: $\{v_1, \ldots, v_g\}$;
defining the set of edges $E^*$ of graph $G^*$ as the set of pairs $(v_i, v_j)$ $(i < j)$; computing the length $\rho(i, j)$ of each edge $(v_i, v_j) \in E^*$ as the **length of the shortest path** between the corresponding nodes in the original graph $G$;
computing the amounts $N_\nu = \sum_{i \in [\nu]} n_i$ $(\nu \in [g])$; $N_0 = 0$; numbering the jobs: jobs located at node $v_\nu$ get indices from the interval $[N_{\nu-1} + 1, N_\nu]_{\mathbb{Z}}$.
(b) Searching for the hamiltonian cycle $(H^*)$ of minimum length $\rho(H^*)$ in network $G^*$.
(c) Computing the lower bound on the optimum for the given instance of problem $ROS^*$-$UET$: $\bar{C} := \rho(H^*) + n$.

***(line 2) Begin of the cycle*** *on* $\Delta := 1, \ldots, m$
Computing the upper bound $L(\Delta) := \bar{C} + \Delta - 1$ on schedule length,
the upper bound $2g - 2 + \Delta$ on the number of stays of any machine
and the upper bound $\hat{s} := m(2g - 2 + \Delta)$ on the total number of stays.

***(line 3) Begin of the cycle*** *on the total number of stays:* $s := m(g + 1), \ldots, \hat{s}$

***(line 4) Begin of the cycle*** *on variants of the total sequence of stays* $T$*;*
specifying a variant of a sequence consisting of $s$ pairs $(\mu(q), \nu(q)) \in [m] \times [g]$.

***(line 5) Computing a "supplementary information" for the given variant of*** $T$
% Compute the number of stays $(s_k)$ of each machine $M_k$ $(k \in [m])$ and the number $(s_{k,\nu})$
% of its stays at each node $v_\nu$ in arrays $s1[1..m]$ and $s2[1..m, 1..g]$; the $T$-route of machine
% $M_k$ and its length are accumulated in sequence $R_T(k)$ (of node indices)
% and in the array $\rho_T[1..m]$, respectively. Initialization:
$s1 \equiv 0$; $s2 \equiv 0$; $\bar{s} := 0$; $\rho_T \equiv 0$; $R_T(k) :=$ **null**, $\forall\, k \in [m]$;
**for** $q := 1, \ldots, s$ **do begin** % scanning the sequence $T$
$\bar{\mu} := \mu(q)$; $\bar{\nu} := \nu(q)$; $t := s2[\bar{\mu}, \bar{\nu}] + 1$; % a one more stay of machine $M_{\bar{\mu}}$
$s2[\bar{\mu}, \bar{\nu}] := t$; % at node $v_{\bar{\nu}}$ is found ($t$ is its *local index*)
$\beta(\bar{\mu}, \bar{\nu}; t) := q$; % the transition from the *local index* $t$ to the *absolute index* is computed
$s1[\bar{\mu}] := s1[\bar{\mu}] + 1$; % the *machine index* of the current stay is computed
$\gamma(\bar{\mu}, s1[\bar{\mu}]) := q$; % and the transition function from the *machine index* to the *absolute* one
$R_T(\bar{\mu}) := R_T(\bar{\mu}) \oplus \bar{\nu}$ % have incremented the $T$-route of machine $M_{\bar{\mu}}$ by node $\bar{\nu}$
**if** $s1[\bar{\mu}] > 1$ **then** $\rho_T(\bar{\mu}) := \rho_T(\bar{\mu}) + \rho(\nu(\gamma(\bar{\mu}, s1[\bar{\mu}] - 1)), \bar{\nu})$ % and incremented its length
**if** $\bar{\nu} > g_{ns}$ **then begin**

$\bar{s} := \bar{s} + 1$; % a new special stay is found; $\bar{s}$ is its *relative index*
$\bar{q}(\bar{s}) := q$; % the transition from the *relative index* to the *absolute* one
$RN(\bar{\mu}, \bar{\nu}; t) := \bar{s}$; % computing the transition function from the *local index* $t$
% of a special stay to the *relative* one
$\kappa(\bar{s}) := s1(\bar{\mu})$; % and from the *relative index* to the *machine index*
**end (if)**
**end (for** $q$**)**.

***(line 6) Verification of the variant of*** $T$ ***on its correctness.*** (The iteration of the cycle on $T$ is terminated ahead of time, if one of the conditions (a)-(e) is unsatisfied.)

***(line 7) Begin of the cycle*** *on variants of function* $\boldsymbol{D : [\bar{s}{+}1] \to \{0, 1, \ldots, 2m\}}$.
Put $D(1) = D(\bar{s} + 1) = 2m$. For other $t \in [2, \bar{s}]_{\mathbb{Z}}$, we choose integral values $D(t) \in [0, 2m]_{\mathbb{Z}}$.

***(line 8) Computing a "supplementary information" for the given variant of function*** $D$
In parallel with specifying the values of function $D$, we compute the (derived from $D$) *segment coordinates* $(SC1(t), SC2(t))$ of each special stay ($t \in [\bar{s}]$):
**for** $t := 1$ **to** $\bar{s}$ **do**
**if** $D(t) < 2m$ **then** $\{SC1(t) := SC1(t-1);\ SC2(t) := SC2(t-1) + D(t)\}$
**else** $\{SC1(t) := t;\ SC2(t) := 0\}$

***(line 9) Begin of the cycle*** *on variants of the relative schedule of special jobs (RSS).*
*The relative schedule* was defined in Section 2 as a set of pairs of *local coordinates* specified for each special operation $O_j^k$. At that, coordinate $LC_1(O_j^k) \in [s_{k,Loc(j)}]$ specifies the *local index* of its *personal stay*, while $LC_2(O_j^k) \in [n_{Loc(j)} + \Delta - 1]$ specifies the *shift* of the **completion time** of operation $O_j^k$ with respect to the **starting time** of its personal stay.

***(line 10) Computing a "supplementary information"*** (such as segment coordinates of special operations and lengthes of special stays).

$A(P, t) \equiv 0$ $(t \in [\bar{s}])$;
**for** $k := 1$ **to** $m$ **do begin**
**for** $j := n_{NC} + 1$ **to** $n$ **do begin**
$t := RN(k, Loc(j), LC_1(O_j^k))$;
$SC_1(O_j^k) := SC1(t);\ SC_2(O_j^k) := SC2(t) + LC_2(O_j^k)$;
$A(P, t) := \max\{A(P, t), LC_2(O_j^k)\}$
**end (for j)**
**end (for k)**

***(line 11) Verification of the variant of*** $RSS$ ***on its correctness***
Verification of the non-simultaneity conditions is performed for pairs of related special operations — separately for each machine and for each special job, by comparing segment coordinates of two operations (by Lemma 3.7). The verification of the current $RSS$ is terminated ahead of time, if one of those conditions fails.

**Step 1: machines.** For each machine $M_k$ ($k \in [m]$), we form the set $X_k$ of pairs of segment coordinates of special operations $O_j^k$ over all special jobs $j \in SJ \doteq \cup_{\nu \in SN} \mathcal{J}(\nu)$. We next organize the pairs in $X_k$ in lexicographically non-decreasing order and compare every two consecutive pairs. Finding two identical pairs means that the non-simultaneity requirement is failed (by the criterion of Lemma 3.7).

**Step 2: jobs.** Perform the same for each job $j \in SJ$.

It is clear that the running time of the described procedures is bounded by a polynomial of $m$ and $g$, which is dominated by the exponent arising while estimating the number of variants of pre-schedule $P$.

***(line 12) Generating the constraints of problem*** $ILP'(P)$ ***for the given variant of pre-schedule*** $P$***. Solving the problem*** $ILP'(P)$***.***
Given a pre-schedule $P$, we form the constraints of problem $ILP'(P)$.
For solving the problem $ILP'(P)$, we apply the algorithm of (Frank and Tardos, 1987; see also Theorem 3.3, p. 27 (Lokshtanov, 2009))

***(line 20) Constructing the local schedules at special nodes*** $v_\nu$ $(n_\nu < m)$
The completion time of each special operation is computed by formula (6), based on the relative schedule (specified in the given pre-schedule $P$) and on the solution obtained of problem $ILP'(P)$.

***(line 21) Constructing the local schedules of jobs at semi-special nodes*** $v_\nu$ $(m \le n_\nu < 2s(\nu))$ ***by implementing Procedure 1***

A local schedule of jobs at a semi-special node $v_\nu$ is constructed by means of the reduction of that scheduling problem to the problem of searching for a proper edge coloring of a bipartite graph in a minimum number of colors (see the details of the reduction in the proof of Lemma 3.8). For solving the latter problem, we use the algorithm from (Cole et al, 2001) with bound $O(mn_\nu \log n_\nu)$ on its running time. Since that procedure is applied in our algorithm only for *semi-special nodes* (where $n_\nu < 2s(\nu)$, by Definition 2.16, see page 52), the total running time of the application of Procedure 1 (over all semi-special nodes) does not exceed $O(ms \log s)$, which, in view of the bound $s \le O(m(m+g))$, is not greater than $O(m^2(m+g)\log mg)$.

***(line 22) Constructing the local schedules of jobs at absolutely-non-special nodes*** $v_\nu$ $(n_\nu \ge 2s(\nu))$ ***by means of Procedure 2***
To construct a feasible schedule at each ANS-node, we apply Procedure 2 from the proof of Lemma 3.9. Due to a quite simple form of the schedule (which exists for any ANS-node), we have a possibility for its compact presentation and efficient computing, as shown in the proof of Lemma 3.9. The total running time of the application of Procedure 2 (over all ANS-nodes) amounts $O(s \log s + ms)$ elementary operations, which does not exceed the total running time of the application of Procedure 1.

4.4. **Justification of the correctness and the optimality of the algorithm. The bound on its running time.** We leave the reader to make sure that in the described above algorithm $\mathcal{A}$ only *elementary operations* are being performed (see Definition 2.4), and that the encoding length of the operands of each such operation (and hence, due to Remark 1 and Definition 2.5, also the *bit running time* of the operation) does not exceed (by the order of magnitude) the amount $|I|$, i.e., the

encoding length of the input of the problem instance $I$ in its *compact encoding*. For example, the encoding length of each temporal parameter in any schedule under consideration (in view of the upper bound on the optimum obtained in Lemma 3.2) does not exceed $\log L(\Delta) \leq \sum_{e\in E^*} \log \rho(e) + \log n + \log m = \underline{|I|} \leq |I|$. Whence, to estimate the *bit running time* of the algorithm, it is sufficient to estimate the **number of elementary operations** performed in the algorithm (which in our case amounts the notion of the "running time" of the algorithm), and then to multiply it by $|I|$.

**Lemma 4.11.** *The bit running time of algorithm $\mathcal{A}$ of solving the $ROS^*$-$UET$ problem can be estimated by the amount $2^{O(g_{sn}m^2 \log m+(m^2+mg)\log mg)} \cdot |I|$, where $m$ is the number of machines, $g$ and $g_{sn}$ are the numbers of all and special nodes of the reduced network $G^*$, respectively, and $|I|$ is the encoding length of the input of the $ROS^*$-$UET$ problem in its compact encoding.*

*Proof.* We remind the reader that by $B(X)$ we denote the number of those variants of values of component $X \in \{\Delta, s, T, D, RSS\}$ of a pre-schedules $P$ that are enumerated in the algorithm; by $\mathcal{T}(X)$ we denote the running time of "treating" a component $X$. (By "treating" a component $X$, we mean computing a supplementary information depending on $X$ and checking the feasibility of the given variant of component $X$). By $\mathcal{T}(ILP)$, we denote the running time of the solution of problem $ILP'(P)$. The running time of the whole algorithm $\mathcal{A}$ includes the running time of the Main part of the algorithm (the *Big Cycle*), denoted by $\mathcal{T}_{MP}$, and of the running time of the Preliminary and Final stages.

The most time consuming part of the *Preliminary stage* is the searching for the shortest hamiltonian cycle in the reduced network $G^*$. The running time of this part, according to Theorem 3.1, can be estimated as $2^{O(g)}$. *The Final stage* ($CP$) consists in the consecutive application of the procedures of constructing feasible local schedules of jobs at each node $v_\nu$, $\nu \in [g]$. As was show in Section 4.3 (***line 21*** and ***line 22***), the running time of this stage is not greater than the polynomial $O(m^2(m+g)\log mg)$, and thus, as will be seen from further bounds, both bounds (on the running time of the Preliminary and of the Final stages) are majorized by the bound on the running time of the Main part. Thus, it remains to estimate the running time of the Main part which can be presented by the following formula:

$$\mathcal{T}_{MP} \leq B(\Delta) \cdot B(s) \cdot B(T)(\mathcal{T}(T) + B(D) \cdot B(RSS) \cdot (\mathcal{T}(RSS) + \mathcal{T}(ILP))) \quad (8)$$

We first estimate the amounts $B(X)$ and $\mathcal{T}(X)$ for each component $X$ of a pre-schedule $P$. Clearly, $B(\Delta) = O(m)$ and $B(s) = O(\hat{s}) \leq O(m(m+g))$. The other amounts $B(X)$ can be taken from Section 2: $B(T) \leq 2^{O(m(m+g)\log mg)}$, $B(D) \leq 2^{O(m(m+g)\log m)}$, $B(RSS) \leq 2^{O(g_{sn}m^2 \log m)}$. As we can see, $B(D) \leq B(T)$.

The time needed for the "treatment" of each variant of component $T$ includes the time needed for the extraction from $T$ the "supplementary information" (including the *transition functions* between various numberings of stays: the *absolute, relative, machine and local* ones; having these functions, we can determine machine routes through the nodes of network $G^*$, as well as the sequence of special stays) and — the time for checking the correctness of the given variant of $T$. As can be seen from the description of (***lines 5,6***) in Section 4.3, the total running time of all those procedures is linear in $s$, or, subject to the constraints on the total number of stays (specified in a given pre-schedule $P$), is polynomial of $m$ and $g$. Thus, it is "majorized" by the exponent value of $B(T)$.

The checking of a variant of $RSS$ on its feasibility reduces to the verification of satisfiability of the non-simultaneity requirement to related operations at each special node $v_\nu$. By Lemma 3.7, to perform such a verification, it is sufficient to compute and compare segment coordinates of related special operations. Quite clear that the running time of these procedures is bounded above by a polynomial function $P_{RSS}(m, g)$ of $m$ and $g$, which provides the same bound on $\mathcal{T}(RSS)$.

Finally, by Lemma 3.6, the running time of the solution of problem $ILP'(P)$ does not exceed

$$\mathcal{T}(ILP) \leq 2^{O(s \log s)}.$$

Substituting to formula (8) the bounds on all its components and multiplying the result by $|I|$, we obtain the final bound on the bit running time of the algorithm:

$$\mathcal{T}(\mathcal{A}) = O(\mathcal{T}_{MP}) \cdot |I| \leq 2^{O(g_{sn} m^2 \log m + s \log s)} \cdot |I| \leq 2^{O(g_{sn} m^2 \log m + m(m+g) \log mg)} \cdot |I| \,\square$$

We are now ready to prove the main theorem.

**Theorem 4.5.** *For any instance of problem ROS*-UET, algorithm $\mathcal{A}$ constructs its optimal schedule. The bound on the running time of the algorithm (derived in Lemma 4.11) is parameterized with respect to the parameter $m + g$, and is linear (for fixed values of $m$ and $g$) in the input length under its compact encoding.*

*Proof.* The algorithm for constructing the optimal schedule for a given instance of problem $ROS^*$-$UET$ is based on the following two facts.
1) Due to Corollary 3.2, the optimum of any instance of the $ROS^*$-$UET$ problem exists and can be represented in the form $C(S^*) = L(\Delta) \doteq \bar{C} + \Delta - 1$, where $\Delta \in [m]$, and $\bar{C}$ is the lower bound on the optimum derived in Proposition 3.2.
2) By Lemma 3.10, a complete feasible schedule $S$ of length $C(S) \leq L(\Delta)$ exists for a given $\Delta \in [m]$, if and only if $Sol'(\Delta) \neq \varnothing$.

Thus, to find the optimum for a given instance of problem $ROS^*$-$UET$, it is sufficient to find the minimum $\Delta \in [m]$ for which $Sol'(\Delta) \neq \varnothing$. To that end, in turn, it is sufficient to look through the values $\Delta \in [m]$ in the increasing order, for each value of $\Delta$ enumerate all pre-schedules $P \in \mathcal{P}(\Delta)$, and for each $P$ to check the existence (or non-existence) of a feasible solution of problem $ILP'(P)$. (For checking this property, the algorithm of **(author?)** [8] can be used.) The very first pre-schedule $P \in \mathcal{P}(\Delta)$ found, for which $Sol'(P) \neq \varnothing$, provides the desired value of the optimum ($C(S^*) = L(\Delta)$). To construct the optimal schedule itself, it is sufficient to take the solution $\langle X, Y \rangle$ found for problem $ILP'(P)$ (and specifying a semi-schedule) and extend it up to a complete schedule $S$ by finding (for each non-special node) a feasible local schedule of jobs located at that node (the schedule should be consistent with the solution $\langle X, Y \rangle$). The existence of such a schedule at each non-special node is guaranteed by Lemma 3.8. For its construction, it is sufficient to use Procedure 1 applied in the proof of Lemma 3.8, while the feasibility of the complete schedule $S$ was shown in the proof of Lemma 3.10.

Thus, the algorithm working by the above scheme guarantees constructing an optimal schedule for any given instance of problem $ROS^*$-$UET$. The only point at which such an algorithm does not meet our requirements is its running time. Indeed, the running time of Procedure 1 at a node $v_\nu$ can be estimated by the amount $O(mn_\nu \log n_\nu)$. At that, the length of the input (in the compact encoding scheme) amounts only $O(\log n_\nu)$, which provides no possibility to bound the running time of the algorithm by a linear function of the input length (for fixed $m$ and $g$).

As shown in Lemma 3.9, the desired property can be obtained via the application of another algorithm (called there "Procedure 2") of constructing a local schedule of jobs at a non-special node. A polynomial running time of that procedure is provided due to quite a simple form of the schedule under construction, thanks to which the time resources needed for the unique presentation of that schedule become comparable with the input length in the compact encoding (accurate to its multiplication by a function of $m$ and $g$). The truth, for the existence of such a feasible schedule, it is required that the non-special node would contain a large enough number of jobs (more exactly, it must be $n_\nu \geq 2s(\nu)$ jobs; we call such nodes "absolutely-non-special", or "ANS-nodes"). As a result, after the application of Procedure 2 for ANS-nodes (and Procedure 1 for the remaining non-special nodes, called "semi-special"), we obtain the desired bound on the running time of the whole algorithm. Theorem 4.5 is proved. □

## 5. Discussion

As we made certain above, the time required for computing the schedule in the form provided by Procedure 2 is bounded by a function of $m$ and $g$ multiplied by the input length in the compact encoding. However, the question arises: **is this information sufficient for a practical realization of this schedule**?

On the one hand, an important feature of this schedule is that all machines should maintain **the same cyclic order** of processing the jobs at each $ANS$-node. On the other hand, **we need not compute a job order** in the algorithm, since **any job numbering** fits our schedule. To our mind, a possible practical implementation of this schedule (at which all machines are able to maintain the same cyclic order of jobs at a given node) could look as follows.

We can assume that there is a *node administrator* who knows the numbering of the objects at his node (at that, he need not know the schedule). In turn, each machine "knows" the index of the job/object at this node that should be processed (by that machine) first, and knows the index of each next job. Every time that a machine is to start a new (next in turn) job, it calls the index of that job to the administrator, and the latter shows where the object with this index is located.

As can be seen, this scheme works perfectly, but requires $n_\nu$ requests by each machine to the node administrator, which is, clearly, **not a polynomial of the encoding length** of the information on the objects at that node (the latter is proportionate to $\log n_\nu$). Is this critical for evaluating the efficiency of our algorithm? We believe not, because:

(a) this number of requests (linear in $n_\nu$) **is inevitable** (minimum possible) regardless of the form of presentation of the schedule, since the machines, even knowing the indices of all jobs that are to be performed, do not know the correspondence between those indices and real objects;

(b) we need not take into account the time spent by the machines on those requests, since this time is spent not in the process of calculating the schedule, but in that of its **practical implementation** (when, already the processing of jobs by each machine takes time linear in the number of jobs).

## 6. Conclusion

This paper presents the first $FPT$-algorithm of constructing a complete optimal schedule for the routing Open Shop problem with unit execution times of operations

(denoted as $ROS\text{-}UET$). The algorithm allows one to build optimal schedules of processing jobs by machines (and find the corresponding machine routes through the nodes of the transportation network) in time linearly dependent on $|I|$ (the encoding length of the input in its compact encoding), provided that all "critical" problem parameters (such as the number of machines and the number of active nodes of the transportation network) are bounded by constants. In fact, a somewhat stronger result is obtained: the polynomial-time solvability of the problem by the algorithm presented also holds for increasing values of critical parameters (it is sufficient to bound each of those parameters by a function not greater than $(\log |I|)^{1/4}$).

It would be interesting to find out the limits of possible extension of the result obtained (namely, of the possibility for constructing an exact $FPT$-algorithm) to more general problems of that type under some (maybe, other) constraints on critical parameters: the number of machines, the number of active nodes of the network, and the execution times of operations. In the present paper, we have investigated the version of constraints on processing times of operations, in which all operations are of the same unit length. As possible variants of weakening the above constraints, the following cases could be considered:
1) the $ROS\text{-}UETMO$ problem ("UET with missed operations"), which allows missing operations of machines on some objects; consequently, each machine may have its "personal" set of active nodes, and therefore, its own optimal route through the network;
2) the $ROS\text{-}BET$ problem (with "bounded execution times"), in which the execution times of all operations are bounded by a constant;
3) the $ROS\text{-}BNDET$ problem ("bounded number of different execution times"), which allows arbitrarily large operation processing times, but the number of **different** operation processing times is bounded by a constant.

Each of the above problems can be generalized by extending its "route component", so that the travel time of a machine through an edge becomes dependent on the machine (which is natural for those problems in which different job executors use different means of transport; in the problem setting due to Zhu & Wilhelm (2006) [19], this corresponds to the case when the setup times of a machine depend not only on the pair of consequent jobs processed by that machine, but also depend on the machine). Such extensions can be marked by "MDTT" ("machine dependent travel times"). For example, the $ROS\text{-}UETMO$ problem will be converted to "ROS-MDTT-UETMO".

Another direction of possible generalization of the results obtained is replacing the classical Open Shop by a *generalized Open Shop* ($GOS$), in which the number of operations on each object is a parameter independent of the number of machines (thereby admitting several operations of one executor on one object).

## References

[1] A. Allahverdi, C. Ng, T.C.E. Cheng, M.Y. Kovalyov, *A survey of scheduling problems with setup times or costs*, European Journal of Operational Research, **187**:3 (2008), 985–1032; doi: 10.1016/j.ejor.2006.06.060 MR2378326

[2] R. van Bevern, A.V. Pyatkin, *Completing partial schedules for Open Shop with unit processing times and routing*, In: Proceedings of the 11th International Computer Science Symposium in Russia (CSR'16), Springer, Lecture Notes

in Computer Science, **9691** (2016), 73–87; doi: 10.1007/978-3-319-34171-2_6; MR3533846

[3] R. van Bevern, R. Niedermeier, M. Sorge, M. Weller, *The complexity of arc routing problems*, In: A. Corberan and G. Laporte (eds.) *Arc Routing: Problems, Methods, and Applications*, MOS-SIAM Series on Optimization, **20** (2014), SIAM-MOS, 19–52; doi: 10.1137/1.9781611973679.ch2

[4] R. van Bevern, C. Komusiewicz, M. Sorge, *A parameterized approximation algorithm for the mixed and windy capacitated arc routing problem: Theory and experiments*, Networks, **70**:3 (2017), 262–278; doi: 10.1002/net.21742; MR3702528

[5] H.J. Böckenhauer, J. Hromkovič, J. Kneis, J. Kupke, *The parameterized approximability of TSP with deadlines*, Theory of Computing Systems, **41**:3 (2007), 431–444; doi: 10.1007/s00224-007-1347-x; MR2352540

[6] R. Cole, K. Ost, S. Schirra, *Edge-coloring bipartite multigraphs in $O(E \log D)$ time*, Combinatorica, **21**:1 (2001), 5–12; doi: 10.1007/s004930170002; MR1805711

[7] M. Cygan, F.V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh, *Parameterized Algorithms*, Springer, 2015; doi: 10.1007/978-3-319-21275-3; MR3380745

[8] A. Frank, E. Tardos, *An application of simultaneous diophantine approximation in combinatorial optimization*, Combinatorica, **7**:1 (1987), 49–65; doi: 10.1007/BF02579200; MR0905151

[9] T. Gonzalez, S. Sahni, *Open shop scheduling to minimize finish time*, Journal of the ACM, **23**:4 (1976), 665–679; doi: 10.1145/321978.321985; MR0429089

[10] G. Gutin, M. Jones, M. Wahlström, *The mixed chinese postman problem parameterized by pathwidth and treedepth*, SIAM Journal on Discrete Mathematics, **30**:4 (2016), 2177–2205; doi: 10.1137/15M1034337; MR3576563

[11] G. Gutin, M. Jones, B. Sheng, *Parameterized complexity of the $k$-arc chinese postman problem*, Journal of Computer and System Sciences, **84** (2017), 107–119; doi: 10.1016/j.jcss.2016.07.006; MR3570171

[12] G. Gutin, M. Wahlström, A. Yeo, *Rural Postman parameterized by the number of components of required edges*, Journal of Computer and System Sciences, **83**:1 (2017), 121–131; doi: 10.1016/j.jcss.2016.06.001; MR3546865

[13] P.N. Klein, D. Marx, *A subexponential parameterized algorithm for Subset TSP on planar graphs*, In: Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'14), Society for Industrial and Applied Mathematics, (2014), 1812–1830; doi: 10.1137/1.9781611973402.131; MR3376491

[14] E. Lawler, J. Lenstra, A. Rinnooy Kan, D. Shmoys, *Sequencing and scheduling: algorithms and complexity*, In: Handbooks in operations research and management science, Logistics of production and inventory, **4**, North Holland, Amsterdam, (1993), 445–522; doi: 10.1016/S0927-0507(05)80189-6

[15] D. Lokshtanov, *New methods in parameterized algorithms and complexity*, PhD thesis, University of Bergen, Norway, (2009).

[16] M. Mnich, R. van Bevern, *Parameterized complexity of machine scheduling: 15 open problems*, Computers and Operations Research, (2018), in press; doi: 10.1016/j.cor.2018.07.020

[17] C.H. Papadimitriou, K. Steiglitz, *Combinatorial optimization: algorithms and complexity*, Dover Publications, Inc., Mineola. New York, (1998). MR1637890

[18] D. Williamson, L. Hall, J. Hoogeveen, C. Hurkens, J. Lenstra, S. Sevast'janov, D. Shmoys, *Short shop schedules*, Operations Research, **45**:2 (1997), 288–294; doi: 10.1287/opre.45.2.288; MR1644998

[19] X. Zhu, W.E. Wilhelm, *Scheduling and lot sizing with sequence-dependent setup: A literature review*, IIE Transactions, **38**:11 (2006), 987–1007; doi: 10.1080/07408170600559706

René A. van Bevern
Novosibirsk National Research University,
1, str. Pirogova,
Novosibirsk, 630090, Russia
*E-mail address*: rvb@nsu.ru

Artem V. Pyatkin
Sobolev Institute of Mathematics,
4, pr. Koptyuga,
Novosibirsk, 630090, Russia
*E-mail address*: artem@math.nsc.ru

Sergey Sevastyanov
Sobolev Institute of Mathematics,
4, pr. Koptyuga,
Novosibirsk, 630090, Russia
*E-mail address*: seva@math.nsc.ru