S©MR ISSN 1813-3304

СИБИРСКИЕ ЭЛЕКТРОННЫЕ
МАТЕМАТИЧЕСКИЕ ИЗВЕСТИЯ

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

Том 12, стр. 144–144 (2015) УДК 519.854.2
DOI 10.17377/semi.2015.12.xxx MSC 90B35

APPROXIMATE SOLUTION OF THE JOB SHOP PROBLEM BY MEANS OF THE COMPACT VECTOR SUMMATION WITHIN A BOUNDED CONVEX SET IN $\mathbb{R}^d$

SERGEY SEVASTYANOV

Abstract. The first fully polynomial-time approximation algorithm for solving the Job Shop problem in its most general form with theoretically guaranteed a priori accuracy bounds was published in (Sevast'yanov, 1984). The approach used in that paper for solving the Job Shop problem, based on the method of compact vector summation in d-dimensional space, was further developed in (Sevast'yanov, 1986). For any fixed job shop, those algorithms provided an asymptotic optimality of the solutions, under the unlimited growth of size of the batch of jobs supplied at the problem input. Later on, two other approximation algorithms for this problem with ratio performance guarantees appeared in (Shmoys et al., 1994). The first algorithm provided (in polynomial time) a poly-logarithmic approximation in terms of two parameters: the number of machines (m) and the number of operations per job (μ). The second algorithm guaranteed a $(2 + \varepsilon)$-approximation for any fixed $\varepsilon > 0$ and was polynomial-time for any fixed values of m and μ. Finally, in (Jansen et al., 2003), a PTAS was elaborated for the Job Shop problem, which was polynomial-time under the assumption that m and μ are fixed. All three above mentioned algorithms were heavily based on the algorithm from (Sevast'yanov, 1986) and were presented as an "improvement" of the latter. In the present paper, a comparative analysis of the quality of four known for today theoretical algorithms for approximate solution of the Job Shop problem (namely, of the algorithms from (Sevast'yanov, 1986), (Shmoys et al., 1994), and (Jansen et al., 2003)) is carried out. It

Sevastyanov, S.V., Approximate solution of the job shop problem by means of the compact vector summation in a space with asymmetric norm.

The work is supported by the program of fundamental scientific researches of the SB RAS (project 0314-2019-0014) .

Received January, 1, 2015, published March, 1, 2015.

is shown that on the set of instances of practical size, the best to date by the accuracy/efficiency criteria is the algorithm from (Sevast'yanov, 1986), while the second algorithm from (Shmoys et al., 1994) and the PTAS from (Jansen et al., 2003) are not capable of obtaining a solution 10 times worse than optimal in any physically observable time even for comparatively small instances of the problem (and even with the help of the Super-Computer System of the whole Universe). The paper also presents an improved scheme of the algorithm from (Sevast'yanov, 1986), which gives a better bound on the absolute error with the same bound on the running time, which is the first positive result in this direction since 1986. Detailed schemes of all procedures used in the algorithm are given (in order to facilitate its programming). In addition, the paper shows how both algorithms from (Shmoys et al., 1994) can be transformed so that their running time becomes linear in the main problem parameter, the number of jobs n.

Keywords: scheduling, job shop problem, makespan, approximation, polynomial time algorithm.

1. Introduction

In the 1970s, a tight connection was discovered between multistage scheduling problems of the *Flow Shop* type and geometric problems of finding an order of summation for a given finite family of vectors from $\mathbb{R}^d$ within a ball (of some norm defined in $\mathbb{R}^d$) of minimum radius[1]. This connection enabled one to find efficiently approximate solutions for scheduling problems with *a priori* guaranteed accuracy bounds by reducing these problems (approximately) to the corresponding vector summation problems, provided that efficient methods with theoretical performance guarantees had already been developed for the latter problems.

Later on, a similar connection between scheduling problems and vector summation problems was found for many other multistage scheduling problems to the minimum makespan, such as: problems with several different routes of jobs through machines [1, 2, 6, 11, 15, 16], problems with non-fixed routes [9, 22–25], production line problems [3], flexible flow shop problem [15, 20], a.o., which emphasized the importance of vector summation methods for scheduling theory.

As a result, a wide range of geometric problems on finding orders of vector summation within various (either fixed or optimized) domains of finite-dimensional space appeared in the scope of our interests. So, developing efficient methods for approximate solution of such problems became topical. For a more detailed acquaintance with results in this area, we refer the reader to two reviews [21, 25]. Now we only note that beyond the **approximate solution** of scheduling problems, these geometric methods enable one to find **optimal solutions** for wide **polynomially solvable classes of instances** of the *Open Shop* problem [9, 22–25].

By all accounts, one of the most difficult multistage scheduling problems for (not only exact, but also approximate) solution is the Job Shop problem, in which each job may have its own, individual route through machines of an arbitrarily large length. The number of operations per job is a parameter independent on the

[1]For the first time, these results were mentioned by Sevastyanov in June of 1974, in Proceedings of the 3rd All Union Conference on Problems in Theoretical Cybernetics [12]. The relationship between scheduling and vector summation problems was then independently discovered by Belov and Stolin and published in December of 1974 in [4].

number of machines (each machine can be used multiple times in the job route). This m-machine n-job problem with the minimum makespan objective is normally denoted as $\langle J \,||\, C_{\max}\rangle$. It can be formulated as follows. (Here and then, $[k]$ will denote the set of integers $\{1, 2, \dots, k\}$, where $[k] = \varnothing$ for $k = 0$.)

Job Shop problem. We are given a set of jobs $\{J_1, \dots, J_n\}$ that should be processed on a given set of dedicated machines $\{M_1, \dots, M_m\}$. Each job J_j $(j \in [n])$ consists of μ_j *operations* $\{O_{1,j}, \dots, O_{\mu_j,j}\}$ that must be processed in this order. (This means that the next operation of a job cannot start until the previous one has been completed.) Each operation $O_{\eta,j}$ should be processed on a certain machine $M(O_{\eta,j})$ within a time interval of length $p_{\eta,j}$, without interruption. The processing of different operations on the same machine should not overlap in time. Among the set of *feasible schedules* (which meet the above requirements), we wish to find a schedule $S = \{s_{\eta,j} \,|\, O_{\eta,j} \in \mathcal{O}\}$ (where $s_{\eta,j}$ is the starting time of operation $O_{\eta,j}$, and $\mathcal{O}$ denotes the set of all operations) that minimizes the *maximum job completion time* $C_{\max}(S) \doteq \max\{s_{\eta,j} + p_{\eta,j} \,|\, O_{\eta,j} \in \mathcal{O}\}$.

As one can see, the above problem setting stipulates a certain route of each job through machines. As was claimed in [5], such a route cannot be an arbitrary one: "In literature of job shop scheduling, *machine repetition* is usually **not allowed**, *i.e.*, consecutive operations of the same job must always be assigned to different machines. We follow this convention unless stated otherwise."

Apparently, accepting such a convention for the "norm" in the formulation of the Job Shop problem could be explained by the difficulty of solving the problem in the general case. Yet, we should note that such a restriction on the problem setting can facilitate its exact solution in very special cases only (such as the case of two machines) and is useless in more general cases. As for the approximate solution of this problem, as we will see it later, the most significant **positive results** in this direction (due to Sevastyanov [17, 18], Shmoys, Stein and Wein [26], and Jansen, Solis-Oba, and Sviridenko [8]) have been obtained for the Job Shop problem **in the setting presented above**, i.e., **without** the artificial constraint on job routes accepted in Chen, Potts, and Woeginger [5] for the "norm".

Moreover, the problem without such a restriction enables us to consider interesting models with *controllable interruptions*, in which interruptions of each operation are allowed only at certain relative time moments specified by certain (admitted) values of the volume of the already completed part of the operation. To cope with such a problem, it suffices (1) for each operation to cut down its processing time interval at given relative time moments (specifying "allowed interruptions" of the operation), thereby obtaining several "smaller operations" to be sequentially executed **on the same machine**, and (2) to solve the resulting Job Shop problem without interruptions and **without the above restriction on machine repetition**. Such models (with controllable interruptions) generalize both the models **with preemption**, and the models **without preemption**.

Thus, we will consider the Job Shop problem **without** the artificial constraint on machine repetition in job routes (since the latter really limits the generality of our problem) and will denote it as $\langle J \,||\, C_{\max}\rangle$. For denoting the special case of this problem with the "bun" on machine repetition in any job route, we suggest to use notation "no rep". Note also that this restriction limits not the set of feasible solutions of the problem (related to **the problem output**), but rather **the set of its inputs**, thereby determining the problem we solve: either the problem in its general form or a certain special case of it. That is why I propose to place such a definition of the problem to be solved (listing the restrictions/extensions of the set of inputs) **in the first field** of the three-field system we use, leaving the second field for recording the restrictions (or, conversely, extensions) of the set of feasible solutions of the problem. Thus, the above mentioned special case should be denoted as $\langle J, \text{no rep} \,||\, C_{\max}\rangle$.

The difficulty of the exact solution of the Job Shop problem is confirmed by a number of results establishing that some very special cases of it (all of which are "no rep") turn out to be NP-hard. For example, these are the problems $\langle J2, \text{no rep}, \mu_j \leq 3 \,||\, C_{\max}\rangle$ and $\langle J3, \text{no rep}, \mu_j \leq 2 \,||\, C_{\max}\rangle$ [10] (i.e., the problem with two machines and at most three operations per job and the problem with three machines and at most two operations per job), as well as problem $\langle J3, \text{no rep}, n \leq 3 \,||\, C_{\max}\rangle$ [27] (with three machines and at most three jobs). Meanwhile, such a special case of $\langle J3, \text{no rep}, \mu_j \leq 3 \,||\, C_{\max}\rangle$ problem as $\langle F3 \,||\, C_{\max}\rangle$ (when all jobs have the same route $(1, 2, 3)$ through machines) is strongly NP-hard [7].

As far as an approximate solution of this problem is concerned, apparently, the first efficient algorithms with theoretical performance guarantees for solving the Job Shop problem in its most general form were presented in (Sevast'yanov, 1984, 1986). The polynomial-time algorithms described in those two papers guaranteed finding schedules satisfying *a priori* bounds on the **absolute error** in the form:

$$C_{\max}(S) \leq L_{\max} + \varphi(m, \mu) p_{\max}, \tag{1}$$

where $L_{\max}$ is the maximum machine load (which, clearly, is a lower bound on the optimum), $\mu \doteq \max_j \mu_j$ is the maximum number of operations per job, $\varphi(m, \mu)$ is a polynomial on m and μ, and $p_{\max} \doteq \max_{\eta,j} p_{\eta,j}$ is the maximum processing time of an operation.[2] As one can see, the absolute error $C_{\max}(S) - L_{\max} \leq \varphi(m, \mu) p_{\max}$ of the approximate solution does not depend on the number of jobs. Under the assumption that the values of parameters m, μ, and $p_{\max}$ are bounded by constants, while the number of jobs n grows without limit (and so, $p_{\max}$ becomes arbitrarily small with respect to amounts $P \doteq \sum_{\eta,j} p_{\eta,j}$ and $L_{\max} \geq P/m$), the solution found becomes **asymptotically optimal**.

It should be noted that the assumption made in [17, 18] about the limited growth of the quantities m, μ, and $p_{\max}$ in parallel with the growth of the number of jobs n has got a practical meaning. Assuming that the range of products manufactured in the job shop remains more or less constant over a long period of time, we can calculate the maximum values of quantities μ and $p_{\max}$ for this range of products and treat them as constants over time. The maximum possible number of machines (m) placed in the job shop is also limited by the size of the job shop (i.e., by a constant independent of time). As for the number of jobs n, this parameter obviously grows proportionally to the duration (T) of the *planning horizon*[3] of the shop activities. Under these conditions, the algorithms described in [17, 18] provide good practical solutions that become (theoretically) asymptotically optimal with unlimited growth of T.

Based on the algorithm from (Sevast'yanov, 1986), Shmoys et al. (1994) designed two approximation algorithms for the Job Shop problem with **ratio performance**

[2]The algorithms presented in those two papers differ in the orders of their polynomials $\varphi(m, \mu)$ and in bounds on their running time.

[3]*Planning horizon* is the time interval $[0, T]$ during which it is required/wanted/planned to perform a **specified list** of **jobs**.

guarantees. One of them provided a poly-log approximation in terms of **variable parameters** m and μ and was purely polynomial-time. To construct an approximate schedule, they divide the entire set of jobs into "small" jobs (all operations of which have a length not greater than $L_{\max}/(2m\mu^3)$) and the remaining "large" jobs, the number of which (n') does not exceed $2m^2\mu^3$. For "small" jobs, they use the algorithm from (Sevast'yanov, 1986) (with running time $O(m^2\mu^2n^2)$) which constructs a schedule S_S with length that meets bound (1), where instead of the function $\varphi(m,\mu)$ the authors of (Shmoys et al., 1994) use its upper bound $2m\mu^3$. As a result, from (1) and from the upper bound on the lengths of operations of "small" jobs, we obtain the bound $C_{\max}(S_S) \le 2L_{\max}$. For "large" jobs, Shmoys et al. first transform the durations of their operations (by switching to a new time unit equal to $\delta \doteq p_{\max}/|\mathcal{O}'|$, where $\mathcal{O}'$ is the set of operations of "large" jobs), and then rounding them down to the nearest integer. Thus, the new durations ($p'_{\eta,j} \doteq \lfloor \frac{p_{\eta,j}|\mathcal{O}'|}{p_{\max}} \rfloor$) of operations are integers and do not exceed $|\mathcal{O}'| \le n'\mu$. After that, for the instance consisting of all large jobs with transformed processing times, a randomized polynomial-time algorithm is applied, which, with high probability, yields its schedule S'_2 of length

$$C_{\max}(S'_2) \le O\left((L'_{\max} + P'_{\max})\frac{\log(n'\mu)}{\log\log(n'\mu)}\log p'_{\max}\right), \tag{2}$$

where $p'_{\max} \doteq \max_{\{\eta,j\}} p'_{\eta,j}$ is bounded from above by $n'\mu$, while $L'_{\max}$ and $P'_{\max} \doteq \max_j \sum_{\eta\in[\mu_j]} p'_{\eta,j}$ are the maximum machine load and the maximum job length measured in the new time units. Keeping in mind that functions $x/\log x$ and $\log x$ are monotonically increasing, we may replace $p'_{\max}$ and n' in (2) by their upper bounds (polynomial in m and μ) and obtain the bound:

$$C_{\max}(S'_2) \le O\left((L'_{\max} + P'_{\max})\frac{\log^2(m\mu)}{\log\log(m\mu)}\right).$$

This inequality, clearly, remains valid after returning to the original time units, because all time-dependent parameters (such as processing times $p'_{\eta,j}$, schedule length $C_{\max}(S'_2)$ and amounts $L'_{\max}, P'_{\max}$) are multiplied by the same number δ, providing (in new terms) the inequality

$$C_{\max}(S''_2) \le O\left((L''_{\max} + P''_{\max})\frac{\log^2(m\mu)}{\log\log(m\mu)}\right). \tag{3}$$

Next, returning to the original instance of large jobs (with operation lengths $p_{\eta,j} \ge p''_{\eta,j}$) lengthens each operation by at most δ and increases the length of each path in the precedence graph of operations in schedule S''_2 by no more than $\delta\cdot|\mathcal{O}'| = p_{\max}$, while $L''_{\max}$ and $P''_{\max}$ in the right part of (3) grow up to at most $L_{\max}$ and $P_{\max}$. As a result, the authors obtain a schedule S_2 for large jobs which, with high probability, meets the bound:

$$C_{\max}(S_2) \le C_{\max}(S''_2) + p_{\max} \le O\left((L_{\max} + P_{\max})\frac{\log^2(m\mu)}{\log\log(m\mu)}\right). \tag{4}$$

A subsequent derandomization of this schedule yields (in polynomial time) a deterministic schedule S_L for large jobs with bound

$$C_{\max}(S_L) \le O(\log^2(m\mu)C^*_{\max}).$$

Finally, consecutively connecting schedules S_S and S_L for small and large jobs, they obtain schedule S with the relative accuracy bound

$$C_{\max}(S)/C^*_{\max} \le O(\log^2(m\mu)). \tag{5}$$

Since the algorithm for large jobs is polynomial-time, and the number of large jobs (n') is bounded above by $2m^2\mu^3$, the overall running time of their algorithm can be estimated as $O(n^2m^2\mu^2 + P(m,\mu))$, where $P(m,\mu)$ is a polynomial of m and μ.

The **second approximation algorithm from (Shmoys et al., 1994)** provided a $(2+\varepsilon)$-approximation for any fixed $\varepsilon > 0$ in polynomial ($O(n^2)$) time under assumption that parameters m and μ are fixed. Again, they use the idea of dividing the whole set of jobs into two subsets of "small" and "large" jobs, yet with the "watershed" for lengths of operations at a different value $\frac{\varepsilon L_{\max}}{2m\mu^3}$. The application of the algorithm by (Sevast'yanov, 1986) to small jobs yields a schedule S_S that meets bound (1) with $\varphi(m,\mu) = 2m\mu^3$ and $p_{\max} = \frac{\varepsilon L_{\max}}{2m\mu^3}$, which yields the bound $C_{\max}(S_S) \le L_{\max} + \varepsilon L_{\max} \le (1+\varepsilon)C^*_{\max}$. For "large" jobs (the number of which is not greater than $2m^2\mu^3/\varepsilon$, and for fixed m,μ, and ε is fixed), they find (**by any way**) an **optimal schedule** S_L of length $C_{\max}(S_L) \le C^*_{\max}$, which yields the desired accuracy bound for the concatenation of two schedules:

$$C_{\max}(S_S \oplus S_L) \le (2+\varepsilon)C^*_{\max}. \tag{6}$$

The **PTAS presented by (Jansen et al., 2003)** for the general Job Shop problem $\langle J\,||\,C_{\max}\rangle$ in the case of **any fixed** m **and** μ is a development of the above result (i.e., of the $O(n^2)$-time $(2+\varepsilon)$-approximation algorithm for problem $\langle J\,||\,C_{\max}\rangle$ with any fixed m,μ, and $\varepsilon > 0$). They described an $O(n)$-time $(1+\varepsilon)$-approximation algorithm for any fixed $\varepsilon \in (0,1)$ and any fixed m and μ. This time, the authors divided the set of all jobs into three subsets: of "big", "small", and "tiny" jobs. At that, "small" jobs (whose total length is δ-small with respect to $L_{\max}$ and $C^*_{\max}$) are scheduled separately and independently of other jobs, while for the "big" jobs the authors organize a full enumeration of all their "rounded schedules" within a time interval that obviously exceeds the length of the optimal schedule. Each chosen "rounded schedule" for "big" jobs defines time intervals of the "second type", for each of which a subset of machines available for processing the remaining ("tiny") jobs is fixed and known. The target of the linear program then applied is determining suitable lengths of those time intervals and distributing the fragments of "tiny" jobs among those intervals. Finally, the algorithm by (Sevast'yanov, 1986) is separately applied in each "second type" interval to the fragments of "tiny" jobs distributed to it.

As the reader could be convinced from the presented above short description of the approximation algorithms from (Shmoys et al., 1994) and (Jansen et al., 2003), all these algorithms for the Job Shop problem are heavily based on the approximation algorithm by (Sevast'yanov, 1986) and essentially use its bound (1) on the absolute error of the solutions obtained. Meanwhile, no progress can be observed with respect to that absolute accuracy bound since the distant 1986. In the present paper, we make the first step towards a further enhancement of that basic result by improving the absolute accuracy bound guaranteed for the solution obtained by our algorithm. While (Shmoys et al., 1994) roughly estimated the polynomial $\varphi(m,\mu)$ (introduced in (1)) from above in terms of its major term as $2m\mu^3$, we now show that it can be bounded by $m\mu^2(\mu-1)$. To improve the bound, we used two new ideas. One of them is an implementation of the property of the Steinitz family of vectors defined for the original instance of the Job Shop problem to lie within a ball of an asymmetric norm (but with an internal central symmetry), which enables us to apply to these vectors a summation algorithm with a smaller bound on the radius (compared to the algorithms used in (Sevastyanov, 1984, 1986)).

Another idea is to use a large number of additional ("dummy") jobs with zero vector of operation durations. Yet, since an uncontrollable increase in the total number of vectors leads to an increase in the running time of the vector summation algorithm, we have to introduce an upper limit on the number of additional jobs. Thus, the resulting accuracy bound and the bound on the running time of the new algorithm are a kind of a "deal" between two goals: minimum error and minimum running time.

The rest of the paper is structured as follows. Section 2 introduces some necessary notions and notation and presents some known results used in the paper. Section 3 describes the new version of the algorithm for solving the Job Shop problem. At the same time (in order to avoid referring the reader to inaccessible sources and to provide a self-contained description of our algorithm), a sufficiently detailed description of the procedures dealing with families of vectors (and essentially used in our algorithm) is given in Appendices A and B. (This can considerably help the readers interested in an experimental verification of the practical applicability of our method for solving the Job Shop problem with real size instances.) Section 4 presents the proof of the main result of this paper, Theorem 2, where the quality of the new algorithm is estimated.

In addition, Section 5 will demonstrate a simple technique for linearization of the complexity of both algorithms from (Shmoys et al., 1994) with respect to the number of jobs. In terms of the parameters m, μ, ε, and n, the new bounds on their running time will be $O(\mu n + f(m, \mu))$ and $O(\mu n + g(m, \mu, \varepsilon))$, respectively, where $f(m, \mu)$ is a polynomial of m and μ (which is of theoretical interest from the view point of parametric analysis of the complexity of approximate solution of the Job Shop problem), while function $g(m, \mu, \varepsilon)$ exponentially depends on the parameters m, μ, and $1/\varepsilon$. In Section 6, we perform a comparative analysis of four (currently known) approximation algorithms for solving the Job Shop problem with theoretical performance guaranties subject to their applicability for solving practical instances of the problem. The analysis shows an undeniable advantage of our algorithm over other competitors. Moreover, a small instance clearly demonstrates that the second algorithm from (Shmoys et al., 1994) and the PTAS from (Jansen et al., 2003) are absolutely inoperable, neither now nor in any distant future. (We hope that the methodology of assessing the practical applicability of algorithms, presented here, can also be successfully applied for testing the practical applicability of other algorithms of solving discrete problems for which there are theoretically guaranteed bounds on their running time.) Finally, Section 7 contains a brief discussion of the results obtained and formulates further possible research directions.

2. Preliminary results

The notion of a *schedule* is one of the basic notions in our paper. Each schedule defined or constructed in this paper by one of our algorithms represents a certain

solution of the problem under consideration. (Although, some schedules constructed here may be infeasible.) Concerning the definition of schedule, it should be noted that there may be many ways to define this notion (depending on the complexity of the problem setting). But in our case it will be sufficient to define it as a **family of non-negative starting times of all operations**.

Definition 1. A feasible schedule for a given set of jobs is called an *active schedule*, if the starting time of no operation can be decreased without either violating the feasibility of the schedule or without increasing the starting time of some other operation.

It is rather evident that in scheduling problems with the minimum makespan objective (as is in our case) we can always restrict the set of the solutions (to be considered) by the set of active schedules only.

It is well known that when for a given scheduling problem, the feasibility of a schedule is determined **only by some precedence constraints** (specified by a directed network $G = (V, E)$ of type "nodes — tasks"), and network G contains no positive-length cycles, then there is a single active schedule (which is, thereby, the optimal one), and this schedule can be found very efficiently, in $O(|V| \cdot |E|)$ time. (When G contains no cycles at all, this can be done even faster, in $O(|E|)$ time.) Alternatively, when G contains positive-length cycles (which can be established also in $O(|V| \cdot |E|)$ time), the given problem instance **has no feasible solutions**.

When, beyond precedence constraints, the problem under consideration has got also renewable resource constraints, the problem normally becomes NP-hard, and the set of all active schedules may have an exponential size. Such a situation takes place for the Job Shop problem (because each machine can be treated as a resource of renewable type), and that is why the problem is so hard for its optimal solution. In this paper, we overcome this obstacle by choosing (at Stage 7 of our algorithm) for each machine a certain processing order of its operations. (We thereby eliminate all resource constraints in the problem and come to the "favorable" situation, when there are only precedence constraints.) Our only concern in this case is to check that the precedence constraints we added (and embedded in network G) together with the original arcs of network G do not form cycles of positive length. And this is done at Stage 7.

In what follows, we will assume that the input jobs of the Job Shop problem are specified by a **list** (in arbitrary order), and the information about the operations of each job J_j (the duration $p_{\eta,j}$ of each operation $O_{\eta,j} \in \mathcal{O}_j$ and machine $M(O_{\eta,j})$ on which it should be processed) is also specified in a list, according to the processing order of job operations. In these conditions, two simple approximation results (formulated below in Lemma 1) are valid for the Job Shop problem.

Let $\mathcal{A}_{01}$ be the algorithm that schedules jobs sequentially, one after another (in an arbitrary order), without delay. Clearly, such a schedule S_{01} can be computed in $O(|\mathcal{O}|) \le O(n\mu)$ time.

Let $\mathcal{A}_{02}$ be another algorithm that schedules all operations level-by-level (for levels $\eta \in [\mu]$). To that end, it scans the list of jobs μ times, and at the η-th time (at *Stage* $\eta \in [\mu]$), schedules all operations of the η-th level as follows. We define amounts $\bar{C}^i_\eta$ ($\eta = 0, 1, \dots, \mu;\ i \in [m]$) which mean the maximum completion time over all operations of the η-th level on machine M_i (where $\bar{C}^i_0 \leftarrow 0,\ \forall\ i \in [m]$). At the beginning of Stage η we first put $\bar{C}^i_\eta \leftarrow \bar{C}_{\eta-1} \doteq \max_{i\in[m]} \bar{C}^i_{\eta-1}$. Then, while considering operation $O_{\eta,j}$, we put $s_{\eta,j} \leftarrow \bar{C}^i_\eta$ (for machine $M_i = M(O_{\eta,j})$) and $\bar{C}^i_\eta \leftarrow \bar{C}^i_\eta + p_{\eta,j}$. Thus, the resulting schedule S_{02} consists of μ *layers*, and in each layer $\eta \in [\mu]$ (where all operations of the η-th level are processed), at least one machine works without idle time. The running time of this algorithm can be estimated as $O(|\mathcal{O}| + m\mu) \leq O((n+m)\mu)$.

Lemma 1. *Algorithm $\mathcal{A}_{01}$ finds an m-approximate schedule S_{01} in time $O(n\mu)$. Algorithm $\mathcal{A}_{02}$ finds a $\min\{m,\mu\}$-approximate schedule S_{02} in time $O(n\mu + m\mu)$.*

Proof. First, it should be observed that both schedules S_{01} and S_{02} meet the property: at each time moment during the makespan, at least one operation is being processed. This implies that the lengths of both schedules meet the bound: $C_{\max}(S_x) \leq P \leq mL_{\max}$. Furthermore, since in each layer of schedule S_{02} at least one machine is idles, the length of each layer is not greater than $L_{\max}$, which yields the bound: $C_{\max}(S_{02}) \leq \mu L_{\max}$. Recalling that $L_{\max}$ is the lower bound on the optimum, we get what is required in the lemma. □

The following theorem was proved in [19].

Theorem 1 ([19]). *Let $X = \{x_i \mid i \in [N]\}$ be a given family of N vectors in $\mathbb{R}^d$, s.t. $\sum_{x_i \in X} x_i = 0$, and let $H(X) \doteq \mathrm{conv}\{X\}$ be its convex hull, $a \in \mathbb{R}^d$ be an arbitrarily chosen vector in $\mathbb{R}^d$, and $H_a(X) \doteq \mathrm{conv}\{0, a - H(X)/d\}$. Then a permutation $\pi = (\pi_1, \ldots, \pi_N)$ of indices $\{1, \ldots, N\}$ defining an order of summing the vectors of family X can be found by algorithm $\mathcal{A}_1(d, a, N, X; \pi)$ in $O(N^2 d^2)$ time, such that*

$$\sum_{i \in [k]} x_{\pi_i} \in (d-1)H(X) + H_a(X), \ \forall\ k \in [N]. \tag{7}$$

(For the description of algorithm $\mathcal{A}_1$, we refer the reader to Appendix B.)

3. Approximation algorithm $\mathcal{A}(\gamma)$ for the Job Shop problem

Algorithm $\mathcal{A}(\gamma)$ consists of a *Preliminary Stage* and 7 *basic stages.* The first three basic stages are destined to transform the given problem instance to a more suitable form. The other four stages are targeted to construct the desired schedule. First, we will clarify the destination of each stage. Then we start studying the operation of the algorithm stage by stage.

At the **Preliminary Stage**, we compute the *maximum processing time of an operation* ($p_{\max} \doteq \max_{\eta,j} p_{\eta,j}$), the total workload $L^i \doteq \sum_{O_{\eta,j} \in \mathcal{O}^i} p_{\eta,j}$ of each machine M_i (where $\mathcal{O}^i$ is the set of all operations to be processed on machine M_i $(i \in [m])$), and the *maximum machine load* $L_{\max} \doteq \max_{i \in [m]} L^i$ (being, clearly, a lower bound on the optimum of the Job Shop problem).

At **Stage 1**, a *unification of job routes* through machines is performed, thereby eliminating the main obstacle of the Job Shop problem — the differences in routes of different jobs! To that end, we make two job transformations.

(1) We equalize the number of operations of all jobs $j \in [n]$ up to the maximal one ($\mu_j \leftarrow \mu \doteq \max_j \mu_j$) by adding to each job $j \in [n]$ dummy operations of levels $\eta \in (\mu_j, \mu]$. Now, each job has got an operation of each *level* $\eta \in [\mu]$.

(2) For each job $j \in [n]$, we expand the set of its operations of each level $\eta \in [\mu]$ from a single operation (to be processed on a certain machine) to m operations, to be processed (independently of each other and even, maybe, simultaneously) on

m different machines. Thus, each such *wide job* consists of $m\mu$ operations. All m operations of level η of any job j have to follow all operations of the same job of smaller levels. Still, we do remember that each wide job at each level η has **at most one real operation**. All other its operations are "dummy" ones, and initially receive zero processing times.

At **Stage 2**, we *equalize the workloads* $\{L^i\}$ of all machines $\{M_i\}$ by increasing the lengths of some operations (both real and dummy ones) up to at most $p_{\max}$, while keeping the values of $p_{\max}$ and $L_{\max}$ unchanged.

At **Stage 3**, *the set of n wide jobs is extended by N additional zero-length jobs.* All $\widetilde{N} \doteq n+N$ resulting jobs and their operations will then be called *extended* ones.

As will be seen from bound (19) on the length of the schedule constructed by algorithm $\mathcal{A}(\gamma)$ (see page 21), increasing the parameter $\widetilde{N}$ (present in the additive term $2(\mu-1)L_{\max}/\widetilde{N}$ in the bound on schedule length) reduces this bound. At the same time, the running time of the algorithm increases proportionally to the square of $\widetilde{N}$. Thus, the choice of a certain value of this parameter while implementing the algorithm is the result of a "deal" between two incomparable goals: the minimum of schedule length and the minimum of running time.

In the former version of the algorithm, when no new jobs were added to the original family of n jobs (i.e., we used the minimum possible value $\widetilde{N} = n$ of this parameter), the additive term in the bound on schedule length depending on $\widetilde{N}$ was about $2\mu^2 p_{\max}$. In the present paper, we set $\widetilde{N} = 3n$ (tripling the number of jobs) and obtain a value of this term about 1/3 of its maximum size. In principle, we could make this term arbitrarily close to zero by letting $\widetilde{N}$ tend to infinity. But this, clearly, would require too high cost (in terms of the running time).

The reasons for the influence of parameter $\widetilde{N}$ on the bound on schedule length may become clearer to the reader after heaving read the explanation of Stage 6 and the derivation of bound (19) in the proof of Lemma 3.

At **Stage 4**, the Job Shop problem is approximately reduced to the problem of finding such an order of summing the vectors from a given family $X \in \mathbb{R}^{m\mu}$ (such that $|X| = \widetilde{N}$, and their sum is equal to zero) which satisfies inclusions (7) (and exists due to Theorem 1). Each vector $x_j \in X$ corresponds to a certain extended job $j \in [\widetilde{N}]$, and the order (of vectors $\{x_j\}$) found at this stage is used in schedules defined at the subsequent stages of our algorithm as the order of job processing.

At **Stage 5**, we define a preliminary (infeasible) *table-based schedule* $S(\widetilde{T})$, where $\widetilde{T}$ is a $\mu \times \widetilde{N}$ table of integers. Its jth column corresponds to job $j \in [\widetilde{N}]$, and the order of columns is that found at Stage 4. Due to inclusions (7), schedule $S(\widetilde{T})$ meets the property that for each job j all its operations of the same level $\eta \in [\mu]$ are started at "about the same time". However, for some η and some job j, the pair of its consequent operations of levels $(\eta-1)$ and η may violate the order of their processing prescribed in job j (thereby providing an infeasible schedule). This shortcoming of schedule $S(\widetilde{T})$ will be corrected in the next stage.

At **Stage 6**, we first observe a sharper property of schedule $S(\widetilde{T})$ (established in (14)): the completion time of an arbitrary operation $O^i_{\eta,k}(\widetilde{T})$ in this schedule "nearly linearly" depends on the **number k of the column** in which this operation is located in table $\widetilde{T}$. More precisely, the completion time of any operation consists of three additive terms such that each incrementing of the parameter k by 1 increases the first term by a **fixed amount** $L_{\max}/\widetilde{N}$, while the amounts of the other two terms are bounded from above by values **independent of** k. (This property of schedule will be further called a *$\hat{k}$-property.*) Thus, the idea arises that in order to eliminate the infeasibility of schedule $S(\widetilde{T})$, table $\widetilde{T}$ should be transformed into a new table $\widehat{T}_\gamma$ (still maintaining the *$\hat{k}$-property*), so as for each $\eta \in [\mu - 1]$ each operation $\widetilde{O}^{i''}_{\eta+1,j}$ (i.e., an operation of the $(\eta+1)$-th level of an extended job j) was localized (in the new table) **by γ columns farther** (for a large enough γ) than any operation $\widetilde{O}^{i'}_{\eta,j}$ (of the same job and of the η-th level).

To perform this relative shift (of extended operations of level $(\eta+1)$ with respect to the same job operations of level η), we **add to the beginning of each row** $\eta \in [\mu]$ of table $\widetilde{T}$ exactly $\gamma(\eta - 1)$ **extra cells** containing zeros (corresponding to dummy operations, further called *S-dummy* ones). At that, the *$\hat{k}$-property* remains valid due to the choice of the length of each S-dummy operation $\widehat{O}^{i}_{\eta}$ to be equal to the **average length** over all extended operations of machine M_i and of level η.

As was said, the value of γ should be large enough (to guarantee that the amount $\gamma L_{\max}/\widetilde{N}$ outperforms the maximum possible value of the two other terms (in total) in the right part of formula (14)). Ideally, it would be great if the amount $\gamma L_{\max}/\widetilde{N}$ was exactly equal to that maximum, because the increasing of γ leads to increasing the length of schedule $S(\widehat{T}_\gamma)$. Yet the matter is that γ may take only **integral values**. As a consequence, the amount $\gamma L_{\max}/\widetilde{N}$ turns out to be greater than needed to guarantee the feasibility of schedule $S(\widehat{T}_\gamma)$. In other words, we can observe here the familiar phenomenon of increasing the bound on some real-valued amount (in our case, of the k-independent part of terms in formula (14)) calculated in terms of an integral number of some fixed units (in our case, of units equal to $L_{\max}/\widetilde{N}$) by an amount comparable with the chosen unit $(L_{\max}/\widetilde{N})$.

Yet, once we cannot avoid the above phenomenon (forced by the integral nature of the parameter γ), could we at least minimize its effect by decreasing the values of the units chosen for measuring? Trying to answer this question, we observe that it is possible to decrease our "rounding error" (in the bound on schedule length) by **increasing the value of the parameter $\widetilde{N}$**. This explains the **introduction of this parameter** at Stage 3 of our algorithm.

At **Stage 7**, we construct an *active schedule* $S_{\text{act}}(\gamma)$ (see the definition of this notion on page 8) for the set of **original operations only.** To that end, we define a directed network $G(\gamma)$ specifying precedence constraints on the set of original operations (treated as nodes of the network). Since these constraints meet the order of these operations defined in the table-based schedule $S(\widehat{T}_\gamma)$ (constructed at Stage 6 and feasible for certain values of γ), this guarantees (for the same values of γ) the existence of a feasible solution for the task scheduling problem $\mathcal{P}(\gamma)$ with precedence constraints specified by network $G(\gamma)$, and so, the existence and uniqueness of an active schedule for this problem. Moreover, we can prove that network $G(\gamma)$ is acyclic, which implies that for finding such an active schedule can be achieved by a simple algorithm in $O(|E|)$ time. Clearly, the length of the schedule found will meet the theoretical bounds derived on the length of schedule $S(\widehat{T}_\gamma)$ (while its practical values should be, hopefully, much less).

We next proceed with a detailed description of our algorithm at stages.

Stage 1 (unification of job routes through machines). While solving of the Flow Shop problem approximately by the compact vector summation method [13, 22], we limited ourselves by constructing *permutation schedules* only (specified by a single permutation of jobs), which made it possible to interpret the vector summation sequence found for the *Compact Vector Summation problem* (the *CVS-problem*) as a sequence of executing the corresponding jobs **by each machine**. At the same time, in the Job Shop problem we observe a radically different situation: instead of a unified "job flow by machines" (which in the Flow Shop problem is dictated by the technology of job execution on a ***flow line***) we observe here a family of **individual job routes by machines** (all of which may be different). And thus, it would seem that there can be no talk of any "unified job flow".

However, we will refute these doubts by unifying the routes of all jobs through machines in the Job Shop problem, which is the target of Stage 1. To do that, we extend the notion of a "job" by adding extra operations to each job.

Definition 2. Let $\mu \doteq \max_{j\in[n]} \mu_j$. We define a *wide job* $\tilde{J}_j$ as a job consisting of $m\mu$ operations $\{O^i_{\eta,j} \,|\, i \in [m],\ \eta \in [\mu]\}$, where each operation $O^i_{\eta,j}$ (called a *wide operation of the* η*-th level*) is to be processed on machine M_i for $p^i_{\eta,j}$ time units. We impose precedence constraints on the operations of job $\tilde{J}_j$:

$$O^{i'}_{\eta,j} \to O^{i''}_{\eta+1,j}, \quad \forall\ i', i'' \in [m],\ \eta \in [\mu-1],\ j \in [n]. \tag{8}$$

Meanwhile, no restrictions are imposed on the order of processing the operations from $\mathcal{O}_{\eta,j} \doteq \{O^i_{\eta,j} \,|\, i \in [m]\}$ (i.e., on the operations of the same level). Moreover, they can be processed **in parallel** (i.e., independently and, maybe, simultaneously, ignoring the fact of their belonging to the same wide job $\tilde{J}_j$). In this sense, the problem to be solved **extends the Job Shop problem**.

The original problem instance with n jobs is embedded into the model with n wide jobs in an obvious way. Each real operation $O_{\eta,j}$ ($\eta \in [\mu_j]$) of job J_j coincides with the operation $O^i_{\eta,j}$ such that $M_i = M(O_{\eta,j})$ and has the same duration: $p^i_{\eta,j} = p_{\eta,j}$, while the durations of all other (new) operations of job $\tilde{J}_j$ are set to zero. (At Stage 2 these durations will be transformed.)

The feasibility of precedence constraints $O_{\eta,j} \to O_{\eta+1,j}$ for the original operations of job J_j follows from (8). Meanwhile, it can be seen that constraints (8), while being **sufficient** for constructing a feasible schedule for each original job J_j, are obviously redundant. This gives hope that the *a posteriori* accuracy bound computed for the solution obtained by our algorithm for each given instance of the Job Shop problem may appear to be much better than its theoretical *a priori* bound. (This conclusion becomes even more obvious after analyzing our actions being performed in the subsequent stages of this algorithm.)

Stage 2 (equalization of machine workloads). We compute the amounts $p_{\max} \doteq \max_{\{i,\eta,j\}} p^i_{\eta,j}$, $L^i_\eta \doteq \sum_{j\in[n]} p^i_{\eta,j}$, and the *workload* $L^i \doteq \sum_{\eta\in[\mu]} L^i_\eta$ of each machine M_i. Since each machine now performs the same number of operations (μn), we can equalize the workloads of all machines up to the amount $L_{\max} \doteq \max_{i\in[m]}\ L^i$ by increasing arbitrarily the durations of some operations $O^i_{\eta,j}$ up to some amounts $\tilde{p}^i_{\eta,j} \le p_{\max}$. As a result of this alignment of the machine workloads, vectors $\tilde{p}_j \doteq \{\tilde{p}^i_{\eta,j} \,|\, i \in [m],\ \eta \in [\mu]\} \in \mathbb{R}^{m\mu}$ ($j \in [n]$) of operation durations will be obtained for all wide jobs $\{\tilde{J}_j \,|\, j \in [n]\}$, with a total vector $\sum_{j\in[n]} \tilde{p}_j = \tilde{L} \doteq \{\tilde{L}^i_\eta \,|\, i \in [m],\ \eta \in$

$[\mu]\} \in \mathbb{R}^{m\mu}$ such that $\tilde{L}^i_\eta \doteq \sum_{j\in[n]} \tilde{p}^i_{\eta,j}$ and

$$\tilde{L}^{\,i} \doteq \sum_{\eta\in[\mu]} \tilde{L}^i_\eta = L_{\max},\ \forall\, i \in [m], \tag{9}$$

$$\tilde{L}^i_\eta \le np_{\max},\ i \in [m],\ \eta \in [\mu]. \tag{10}$$

Stage 3 (extending the set of wide jobs). We extend the set of n wide jobs by adding N new wide jobs of zero length. (The corresponding vectors $\tilde{p}_j$ for all new jobs are set to zero, which leaves the values of all quantities $\{\tilde{L}^i_\eta,\ \tilde{L}^{\,i}\}$, $L_{\max}$, and $p_{\max}$ unchanged.) Denote by $\widetilde{\mathcal{J}}$ the resulting ("extended") set of jobs; $\widetilde{N} \doteq n+N = |\widetilde{\mathcal{J}}|$. All jobs from $\widetilde{\mathcal{J}}$ and their operations will be referred to as "extended jobs" and "extended operations", and denoted as $\tilde{J}_j$ and $\widetilde{O}^i_{\eta,j}$, respectively. (At that, the last N jobs $\tilde{J}_j \in \widetilde{\mathcal{J}}$ have vectors $\tilde{p}_j = \mathbf{0}$.)

As will be seen later, increasing the number $\widetilde{N}$ leads to reducing the upper bound on schedule length. On the other hand, this naturally increases the running time of the vector summation procedure (used at Stage 4 of algorithm $\mathcal{A}(\gamma)$) and of the algorithm $\mathcal{A}(\gamma)$ at its subsequent stages (while constructing a feasible schedule). This, however, can be treated as a fair price for improving its accuracy bound. We could leave the final decision on the choice of the value of $\widetilde{N}$ to each particular user of our algorithm. However, we will make a definite choice of this value in this paper, in order to obtain definite bounds on the accuracy and the running time of our algorithm, formulated in Theorem 2 (page 22).

Stage 4 (reducing the Job Shop problem to the vector summation problem and applying Theorem 1). Let $d = m\mu$, $B \doteq [0, p_{\max}]^d$ be a d-dimensional cube in $\mathbb{R}^d_+$. Then $\sum_{j\in[\widetilde{N}]} \tilde{p}_j = \tilde{L}$ and $\tilde{p}_j \in B$ for all $j \in [\widetilde{N}]$. If to compute the *average vector* $\bar{p}_{\rm ave} \doteq \tilde{L}/\widetilde{N}$ of family $\{\tilde{p}_j \,|\, j \in [\widetilde{N}]\}$, we can define vectors $p'_j \doteq \tilde{p}_j - \bar{p}_{\rm ave}$ forming a so called *Steinitz family of vectors*, i.e., a vector family $X = \{p'_j \in \mathbb{R}^d \,|\, j \in [\widetilde{N}]\}$ satisfying the properties:

$$\begin{gathered}\textstyle\sum_{j\in[\widetilde{N}]}\ p'_j = 0,\\ p'_j \in B' \doteq B - \bar{p}_{\rm ave}.\end{gathered}$$

This enables us to apply Theorem 1 to vector family X and find a permutation $\pi^* = (\pi^*_1, \ldots, \pi^*_{\widetilde{N}})$ of indices $\{1, \ldots, \widetilde{N}\}$ satisfying the inclusions:

$$\textstyle\sum_{j\in[k]}\ p'_{\pi^*_j} \in (d-1)H(X) + H_a(X) \subseteq (d-1)B' + B'_a,\ \forall\, k \in [\widetilde{N}],$$

where $B'_a \doteq \mathrm{conv}\{0, a - B'/d\}$. Since $-B = B - \mathbf{b}$ for vector $\mathbf{b} \doteq (p_{\max}, \ldots, p_{\max}) \in \mathbb{R}^d$, we have $a - B'/d = a - B/d + \bar{p}_{\rm ave}/d = a + (B - \mathbf{b} + \bar{p}_{\rm ave})/d$. Choosing $a = (\mathbf{b} - \bar{p}_{\rm ave})/d$, we obtain $a - B'/d = B/d \ni 0$, implying the equality $B'_a = B/d$ and the inclusion

$$\textstyle\sum_{j\in[k]}\ \tilde{p}_{\pi^*_j} - k\bar{p}_{\rm ave} \in (d-1)(B - \bar{p}_{\rm ave}) + B/d = \alpha(d)B - (d-1)\bar{p}_{\rm ave}$$

for $\alpha(d) \doteq d - 1 + 1/d$. Thus, we obtain the following

Corollary 1. *Algorithm $\mathcal{A}_1$ from Theorem 1 finds in $O(m^2\mu^2\widetilde{N}^2)$ time a permutation $\pi^* = (\pi^*_1, \dots, \pi^*_{\widetilde{N}})$ of indices $\{1, \dots, \widetilde{N}\}$ such that*

$$\sum_{j\in[k]} \tilde{p}_{\pi^*_j} = (k - d + 1)\bar{p}_{\mathrm{ave}} + \alpha(d)\, \delta_k,\ \forall\ k \in [\widetilde{N}],$$

where $d = m\mu$, $\delta_k = \{\delta^i_{\eta,k} \in [0, p_{\max}] \mid i \in [m],\ \eta \in [\mu]\} \in B$.

In the coordinate form, these relations look like:

$$(11)\qquad \sum_{j\in[k]} \tilde{p}^i_{\eta,\pi^*_j} = \frac{k-d+1}{\widetilde{N}} \tilde{L}^i_\eta + \alpha(d)\, \delta^i_{\eta,k},\ \forall\ i \in [m],\ \eta \in [\mu],\ k \in [\widetilde{N}].$$

Stage 5 (defining a preliminary infeasible schedule $S(\widetilde{T})$).

Definition 3. Let a set $\mathcal{O}'$ of $m\mu N'$ operations (including all *extended operations*, defined at Stage 3, and *S-dummy operations*, to be presented at Stage 6) has to be processed on m machines ($\mu N'$ operations per machine). To define a *table-based schedule* $S(T')$ for processing the operations from $\mathcal{O}'$, we first define a two-dimensional table T' with μ *rows* and $N' \geq \widetilde{N}$ *columns.* Each cell $T'(\eta, j)$ ($\eta \in [\mu]$, $j \in [N']$) of table T' contains an integer value, which is either the index $k \in [\widetilde{N}]$ of an extended job $\tilde{J}_k$, or zero (corresponding to an S-dummy operation). In both cases, row number η specifies level η of the operation. Each row $\eta \in [\mu]$ contains all job indices from 1 to $\widetilde{N}$, where each job index $k \in [\widetilde{N}]$ is presented exactly once. Thus, each row η of T' represents a sequence of $(N' - \widetilde{N})$ zeros and $\widetilde{N}$ pair-wise different indices from 1 to $\widetilde{N}$, and thereby, specifies an order of processing the extended operations of the η-th level **on every machine** (because for each row $\eta \in [\mu]$, these orders will be **identical** in schedule $S(T')$ for all machines M_i, $i \in [m]$). For each machine M_i, all S-dummy operations "of the η-th level" (if any) will have identical durations (equal to the average duration over all extended operations of the η-th level on machine M_i) and will not be distinguished from each other in schedule $S(T')$.

Next, we define *the full sequence $\mathcal{P}(T')$ of elements* (η, j) of table T'. In this sequence, the elements follow column-by-column (in the order $j = 1, \dots, N'$), and for each column, in the increasing order of row indices ($\eta = 1, \dots, \mu$). So,

$$(12)\qquad \mathcal{P}(T') :\ (1, 1), (2, 1), \dots, (\mu, 1), (1, 2), \dots, (\mu, 2), \dots, (1, N'), \dots, (\mu, N').$$

Let $O^i_{\eta,j}(T')$ denote the operation from $\mathcal{O}'$ processed by machine M_i and specified by cell $T'(\eta, j)$ of table T'. Then the operations **on each machine** M_i should be processed in schedule $S(T')$ from time zero and on, without an idle time, in the *universal order* specified by sequence $\mathcal{P}(T')$:

$$O^i_{1,1}(T'), \dots, O^i_{\mu,1}(T'), O^i_{1,2}(T'), \dots, O^i_{\mu,2}(T'), \dots, O^i_{1,N'}(T'), \dots, O^i_{\mu,N'}(T').$$

Denoting for each operation $O^i_{\nu,j}(T') \in \mathcal{O}'$ its duration by $\hat{p}^i_{\nu,j}(T')$, and its completion time in schedule $S(T')$ by $\hat{c}^i_{\nu,j}(T')$, we can derive the equalities:

$$(13)\qquad \hat{c}^i_{\eta,k}(T') = \sum_{\nu=1}^{\eta} \sum_{j\in[k]} \hat{p}^i_{\nu,j}(T') + \sum_{\nu=\eta+1}^{\mu} \sum_{j\in[k-1]} \hat{p}^i_{\nu,j}(T'),\quad \eta \in [\mu],\ k \in [N'].$$

Now let the set $\widetilde{\mathcal{O}}$ of *extended operations* introduced at Stage 3 be taken for the set of operations $\mathcal{O}'$. For table T', we take the $\mu \times \widetilde{N}$ table $\widetilde{T}$ in which all extended operations of the ν-th level ($\nu \in [\mu]$) are presented by the ν-th row and are ordered according to permutation $\pi^* = (\pi_1^*, \dots, \pi_{\widetilde{N}}^*)$ of job indices found at Stage 4, while the operations of column j represent the operations of the extended job $\tilde{J}_{\pi_j^*}$ on any machine M_i. Thus, no zeros (corresponding to S-dummy operations) are in table $\widetilde{T}$, because each cell $\widetilde{T}(\eta, j)$ of table $\widetilde{T}$ contains the value $\pi_j^* \in [\widetilde{N}]$. Then, for the table-based schedule $S(\widetilde{T})$ (see Definition 3 above), by means of (13), (11), and (9) (with $\tilde{L}_\nu^i = \sum_{j\in[\widetilde{N}]} \tilde{p}_{\nu,j}^i$), we can derive the following relations:

$$\begin{aligned} \hat{c}_{\eta,k}^i(\widetilde{T}) &= \frac{k-d+1}{\widetilde{N}} \sum_{\nu=1}^{\eta} \tilde{L}_\nu^i + \frac{k-d}{\widetilde{N}} \sum_{\nu=\eta+1}^{\mu} \tilde{L}_\nu^i + \alpha(d) \sum_{\nu=1}^{\mu} \delta_{\nu,k'(k,\nu,\eta)}^i \\ &= \frac{k-d}{\widetilde{N}} L_{\max} + \frac{1}{\widetilde{N}} \sum_{\nu=1}^{\eta} \tilde{L}_\nu^i + \alpha(d) \sum_{\nu=1}^{\mu} \delta_{\nu,k'(k,\nu,\eta)}^i \end{aligned} \tag{14}$$

which hold for all $i \in [m]$, $\eta \in [\mu]$, $k \in [\widetilde{N}]$ (with $k'(k,\nu,\eta) = k$ for $\nu \le \eta$, and $k'(k,\nu,\eta) = k-1$ for $\nu > \eta$). As one can see, each schedule $S_i(\widetilde{T})$ for the operations from $\widetilde{\mathcal{O}}_i \doteq \{\widetilde{O}_{\eta,j}^i \,|\, \eta \in [\mu],\ j \in [\widetilde{N}]\}$ is feasible, since all operations of each extended job $\tilde{J}_{\pi_j^*}$ ($j \in [\widetilde{N}]$) on machine M_i are localized in column j of table $\widetilde{T}$, and so, are processed in the increasing order of their levels. Yet for the feasibility of the overall schedule $S(\widetilde{T})$, we also need fulfilling the requirements (8) for all pairs $(\widetilde{O}_{\eta,\pi_j^*}^{i'}, \widetilde{O}_{\eta+1,\pi_j^*}^{i''})$ of extended operations of job $\tilde{J}_{\pi_j^*}$ that are to be processed on **different machines**. Since operation $\widetilde{O}_{\eta,\pi_j^*}^{i'}$ is localized in the j-th column of $\widetilde{T}$, its completion time $c_{\eta,\pi_j^*}^{i'}$ coincides with $\hat{c}_{\eta,j}^{i'}(\widetilde{T})$, while the starting time $s_{\eta+1,\pi_j^*}^{i''}$ of operation $\widetilde{O}_{\eta+1,\pi_j^*}^{i''}$ coincides with $\hat{c}_{\eta,j}^{i''}(\widetilde{T})$. Thus, all precedence constraints (8) may be satisfied, if and only if the relations

$$0 \le s_{\eta+1,j}^{i''} - c_{\eta,j}^{i'} = \hat{c}_{\eta,j}^{i''}(\widetilde{T}) - \hat{c}_{\eta,j}^{i'}(\widetilde{T})$$

hold for all $i', i'' \in [m]$, $\eta \in [\mu - 1]$, $j \in [\widetilde{N}]$. Clearly, this is possible only in the ideal (unlikely) case, when for each $\eta \in [\mu-1]$ and $j \in [\widetilde{N}]$, all completion times $\{\hat{c}_{\eta,j}^i(\widetilde{T}) \,|\, i \in [m]\}$ coincide in schedule $S(\widetilde{T})$.

Stage 6 (defining a feasible table-based schedule $S(\widehat{T})$). The solution of the above problem is facilitated by the following **Observation** about the dependence of quantities $\hat{c}_{\eta,k}^i(\widetilde{T})$ on parameter k according to formula (14) (i.e., the dependence of the completion time of an arbitrary operation $O_{\eta,k}^i(\widetilde{T})$ in schedule $S(\widetilde{T})$ on the **number** k **of the column** in which this operation is located in table $\widetilde{T}$): **each incrementing the parameter** k by 1 is accompanied by an increase in the first term on the right-hand side of (14) by a **fixed amount** $L_{\max}/\widetilde{N}$, while the other two terms are bounded from above by values **independent of** k. This property of schedule $S(\widetilde{T})$ (i.e., the "almost linear" growth of the completion time of operation $O_{\eta,k}^i(\widetilde{T})$ depending on the column number k) will be briefly called a *$\hat{k}$-property.*

This Observation provides the following idea of defining a feasible table-based schedule: table $T' = \widetilde{T}$ should be transformed into a new table $\widehat{T}_\gamma$, so as for each $\eta \in [\mu-1]$ each operation $\widetilde{O}_{\eta+1,j}^{i''}$ (i.e., an operation of the $(\eta+1)$-th level of an extended job $\tilde{J}_j$) would be localized in the new table **by** γ **columns farther** (for

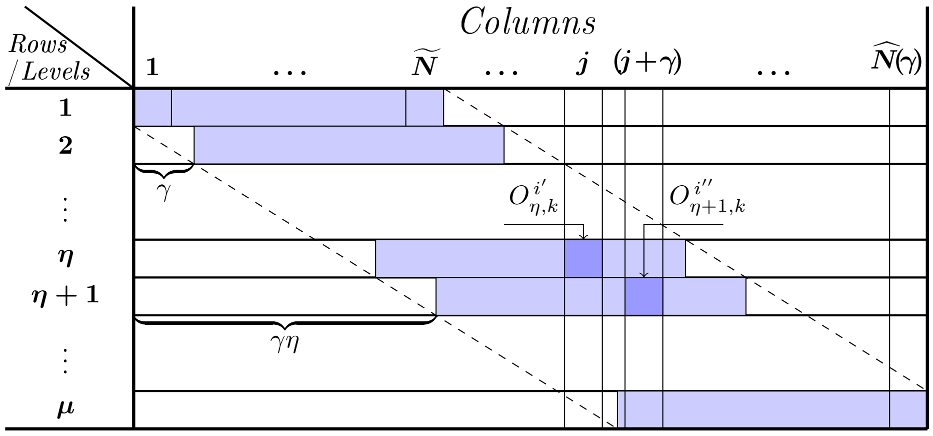

FIG. 1. The construction of Table $\widehat{T}_\gamma$

a large enough γ) than any operation $\widetilde{O}^{i'}_{\eta,j}$ (of the same job) of the η-th level. To provide this relative shifting of extended operations of the $(\eta+1)$-th level with respect to the corresponding operations of the η-th level, we **add to the beginning of each row** $\eta \in [\mu]$ of table $\widetilde{T}_\gamma$ exactly $\gamma(\eta-1)$ **extra cells** containing zeros. For each $i \in [m]$, they will correspond to that many *shifting dummy* (or *S-dummy*, for short) *operations* $\{\widehat{O}^i_\eta\}$ of length $\hat{p}^i_\eta \doteq \tilde{L}^i_\eta/\widetilde{N}$ to be processed in schedule $S(\widehat{T}_\gamma)$ on machine M_i (as described in Definition 3). Furthermore, to keep the balance of the number of operations in all rows of table $\widehat{T}_\gamma$, we also add $\gamma(\mu-\eta)$ extra cells with zeros **to the end of each row** $\eta \in [\mu]$ (see Fig. 1). Thus, each row $\eta \in [\mu]$ of table $\widehat{T}_\gamma$ contains $\widehat{N}(\gamma) \doteq \widetilde{N} + \gamma(\mu-1)$ cells corresponding to operations $\{O^i_{\eta,j}(\widehat{T}_\gamma) \mid j \in [\widehat{N}(\gamma)],\ \eta \in [\mu]\}$ for any $i \in [m]$.

To summarize the above said about the structure of Table $\widehat{T}_\gamma$, we would like to highlight the following its properties.

(a*) Each cell (η, j) of Table $\widehat{T}_\gamma$ corresponds either to m extended operations (of the same extended job $\tilde{J}_{\pi^*_\ell}$, with $\ell \doteq j - \gamma(\eta-1)$, in case $\ell \in [\widetilde{N}]$, when we deal with an e-cell), or to m S-dummy operations (related to no jobs, in case $\ell \notin [\widetilde{N}]$).

(b*) Not every e-cell (η, j) corresponds to *original operations* (being a part of the input instance of the Job Shop problem). The necessary and sufficient conditions for such a correspondence are:

(0*) $\ell \doteq j - \gamma(\eta-1) \in [\widetilde{N}]$ (which means that we deal with an extended job);
(1*) $k \doteq \widehat{T}_\gamma(\eta, j) \in [n]$ (which means that we deal with an original job J_k);
(2*) $\eta \le \mu_k$ (means that we deal with a true level of operations of job J_k).

(c*) If a cell meets all above conditions, then it corresponds to the **unique original operation**, namely, $O^{\hat{i}(\eta,k)}_{\eta,j}(\widehat{T}_\gamma) = O_{\eta,k}$ (the η-th level operation of job J_k to be processed on machine $M_{\hat{i}(\eta,k)} \doteq M(O_{\eta,k})$).

(d*) **Every original operation** is represented in Table $\widehat{T}_\gamma$ **exactly once**.

(e*) For each $i \in [m]$, the sub-sequence of sequence $\mathcal{P}(\widehat{T}_\gamma)$ composed of all cells representing the original operations of machine M_i specifies the order of processing these operations in schedule $S(\widehat{T}_\gamma)$.

This completes the description of Table $\widehat{T}_\gamma$. As will be shown in the proof of Lemma 2 (Sect. 4), assigning to each copy of the S-dummy operation $\widehat{O}^i_\eta$ the same length $\hat{p}^i_\eta = \tilde{L}^i_\eta/\widetilde{N}$ preserves the $\hat{k}$-property for the table-based schedule $S(\widehat{T}_\gamma)$, while a proper choice of the value of γ ensures the feasibility of schedule $S(\widehat{T}_\gamma)$.

Stage 7 (computing the feasible active schedule $S_{\text{act}}(\gamma)$). We define a precedence graph $G(\gamma)$ on the set $\mathcal{O} = \{O_{\eta,k}\}$ of *original operations* ("*o-os*", for short) treated as nodes of the graph. The precedence on the set of operations is defined according to the order of operations of each job (provided by the input data) and to the order of operations on each machine in schedule $S(\widehat{T}_\gamma)$. More precisely, for each o-o $O_{\eta,k}$ we specify at most two outgoing arcs: a job-based arc directed to o-o $O_{\eta+1,k}$ (if $\eta < \mu_k$) and a machine-based arc directed to the o-o $O_{\eta',k'}$, directly succeeding operation $O_{\eta,k}$ on machine $M(O_{\eta,k})$ in schedule $S(\widehat{T}_\gamma)$ (if such an o-o $O_{\eta',k'}$ exists). For the weights of nodes $O_{\eta,k} \in G(\gamma)$ we take the original durations of these operations (valid before performing Stage 2).

Next, we present the most efficient algorithm for implementing this stage in $O(|\mathcal{O}|)$ time. To do this, we load all the initial information about the set $\mathcal{O} = \{O_{\eta,k}\}$ of o-os into an array $A[1..\mu; 1..n]$, so that for any job number $k \in [n]$ and any level $\eta \in [\mu_k]$ of its operation we could add and extract all necessary information on operation $O_{\eta,k}$ in $O(1)$ time. (Such a transformation of the input data can be performed in time linear in the number of o-os.) In addition to the original duration $p[\eta,k] = p_{\eta,k}$ of operation $O_{\eta,k}$ and the number $\hat{i}[\eta,k]$ of the machine executing it, the cell $A[\eta,k]$ will contain: (a) indices $(\eta'[\eta,k], k'[\eta,k])$ of the o-o $O_{\eta'[\eta,k],k'[\eta,k]}$ *directly succeeding* operation $O_{\eta,k}$ on machine $M_{\hat{i}[\eta,k]}$, and (b) the number $ent[\eta,k]$ of arcs entering operation-node $O_{\eta,k}$ in graph $G(\gamma)$. Clearly, $ent[\eta,k] \le 2$ for any node $O_{\eta,k}$, where at most one of two entering arcs may be job-based (it can be obtained from the input data) and at most one may be machine-based (can be obtained from table $\widehat{T}_\gamma$).

All job-based arcs are of the form $(O_{\eta,k}, O_{\eta+1,k})$, and so, provide a straightforward information for graph $G(\gamma)$. In particular, these arcs contribute 1 point to parameters $ent[\eta,k]$ for each node $O_{\eta,k}$ with $\eta \in \{2,\dots,\mu_k\}$ (so that we can put at once: $ent[1,k] \leftarrow 0$ and $ent[\eta,k] \leftarrow 1$ for all $\eta \in \{2,\dots,\mu_k\},\ k \in [n]$).

Meanwhile, to get the information on the machine-based arcs, we have to scan the elements of table $\widehat{T}_\gamma$ according to their *full sequence* $\mathcal{P}(\widehat{T}_\gamma)$ defined in (12) (i.e., column-by-column, and for each column, in the increasing order of row indices). At that, we would like to skip scanning the cells corresponding to S-dummy operations. In other words, we would like to scan **the sub-sequence** $\mathcal{P}'(\widehat{T}_\gamma)$ of sequence $\mathcal{P}(\widehat{T}_\gamma)$ containing only *e-cells* (the cells corresponding to *extended operations*).

This can be done due to two things:

(1) to the special structure of table $\widehat{T}_\gamma$, where in each row $\eta \in [\mu]$ its e-cells follow as a solid segment of length $\widetilde{N}$ (further referred to as an *e-segment*), and for each $\eta \in [\mu - 1]$, e-segment of row $(\eta+1)$ is shifted by γ positions to the right relative to the e-segment of row η (see Fig. 1);

(2) to the choice of the value $\gamma = \gamma^*$ (defined in Lemma 3, page 20).

The following observation can be directly deduced from these two points.

Observation 1. *(a) cells* $(1,1)$ *and* $(\mu, \widehat{N}(\gamma))$ *of table* $\widehat{T}_\gamma$ *are e-cells;*
(b) if (η,k) *is an e-cell and* $(\eta,k) \neq (\mu, \widehat{N}(\gamma))$*, then at least one of three cells* $(\eta+1,k), (\eta,k+1), (\eta+1,k+1)$ *is an e-cell.*

(Here (a) follows from the construction of table $\widehat{T}_\gamma$, and (b) holds due to the inequality $\gamma^* \le \widetilde{N}$, where $\widetilde{N}$ is the length of each e-segment of table $\widehat{T}_\gamma$.)

These properties of table $\widehat{T}_\gamma$ enable us to design the following efficient procedure for scanning all e-cells of table $\widehat{T}_\gamma$ in $O(\mu\widetilde{N})$ time (receiving in parallel the missing information on graph $G(\gamma)$).

```
                 Procedure Scanning (integer: $\gamma$; array of integers: $\pi^*[1..\widetilde{N}]$,
                                   $\eta'[1..\mu;1..n], k'[1..\mu;1..n]$);
% Although the procedure is dedicated to scanning the cells of table $\widehat{T}_\gamma$, the latter
% is not presented at the input of the procedure in an explicit form (for example,
% as an array $\widehat{T}_\gamma[1..\mu;1..\widehat{N}(\gamma)]$). Instead, at the INPUT, we are given an integer $\gamma$
% and the array of integers $\pi^*[1..\widetilde{N}]$ specifying the permutation $\pi^*$ of indices $j \in [\widetilde{N}]$
% of the extended jobs, found at Stage 4 of algorithm $\mathcal{A}(\gamma)$. These two parameters,
% along with $\mu$, provide the complete information on table $\widehat{T}_\gamma$, which enables us
% to do without a direct operation with table $\widehat{T}_\gamma$ when performing the procedure.
% (We will just “keep it in mind”.)
% Formal parameters $\eta'[\eta,k], k'[\eta,k]$ accumulate the indices of the o-o directly suc-
% ceeding $O_{\eta,k}$ on machine $M_{\hat{i}[\eta,k]}$. They represent the OUTPUT of the procedure.
% In the body of procedure Scanning it is assumed that parameters $m, n, \mu, \widehat{N}(\gamma), \mu_k$
% $(k \in [n]), \hat{i}[\eta,k], p_{\eta,k}$ $(k \in [n], \eta \in [\mu_k])$ are global ones.
% Local parameters:
integer: $i, j, k, \eta, \eta^*, \eta^\#, j^\#$;
array of integers: $\eta_c[1..m]$,  $k_c[1..m]$;
% for each $i \in [m]$, they keep the indices $(\eta,k)$ of the last found o-o $O_{\eta,k}$ of machine
% $M_i$ among currently scanned e-cells of table $\widehat{T}_\gamma$.
```

By η^* and $j^\#$, we will denote the indices of the *base row* and the *current column* of table $\widehat{T}_\gamma$, such that in all rows $\eta < \eta^*$ and all columns $j < j^\#$, all e-cells have been already scanned; $(\eta^\#, j^\#)$ will denote the e-cell being currently scanned.

```
BEGIN
  $k_c[i] \leftarrow 0, \ \forall\ i \in [m]$; $(\eta', k')[\eta,k] \leftarrow (0,0), \ \forall\ k \in [n], \eta \in [\mu_k]$;
  $\eta^* \leftarrow 1$; $\eta^\# \leftarrow 1$; $j^\# \leftarrow 1$;
  repeat % scanning the current e-cell $(\eta^\#, j^\#)$ of table $\widehat{T}_\gamma$
    $j \leftarrow j^\# - \gamma(\eta^\# - 1)$; $k \leftarrow \pi^*_j$;
    if $(k \le n)$ & $(\eta^\# \le \mu_k)$ then begin % an o-o $O_{\eta^\#,k}$ is found
      $i \leftarrow \hat{i}(\eta^\#, k)$; if $k_c[i] > 0$ then
        $\{\eta'[\eta_c[i], k_c[i]] \leftarrow \eta^\#$; $k'[\eta_c[i], k_c[i]] \leftarrow k$; $ent[\eta^\#, k] \leftarrow ent[\eta^\#, k] + 1\}$;
      $\eta_c[i] \leftarrow \eta^\#$; $k_c[i] \leftarrow k$
    end; % “if $k \le n$”
    % Next, we search for the next e-cell in sequence $\mathcal{P}(\widehat{T}_\gamma)$
    if $(\eta^\# < \mu)$ & $(j > \gamma)$ then $\eta^\# \leftarrow \eta^\# + 1$ % $(\eta^\#, j^\#)$ will be the next e-cell
```

```
        **else if** $(j^{\#} < \widehat{N}(\gamma))$ **then begin** % The next e-cell is in the next column
          $j^{\#} \leftarrow j^{\#} + 1$; **if** $j^{\#} > \widetilde{N} + \gamma(\eta^* - 1)$      % $(\eta^*, j^{\#})$ **is not an e-cell**
          **then** $\eta^* \leftarrow \eta^* + 1$; % The base row number should be incremented by 1
          $\eta^{\#} \leftarrow \eta^*$
        **end**; % “else”
    **until** $(\eta^{\#}, j^{\#}) = (\mu, \widehat{N}(\gamma))$;
    % Next, we deal with case $(\eta^{\#}, j^{\#}) = (\mu, \widehat{N}(\gamma))$
    $k \leftarrow \pi^*_{\widetilde{N}}$;
    **if** $(k \le n)$ & $(\mu_k = \mu)$ **then begin** % an o-o $O_{\mu,k}$ is found
      $i \leftarrow \hat{i}(\mu, k)$; **if** $k_c[i] > 0$ **then**
        $\{\eta'[\eta_c[i], k_c[i]] \leftarrow \mu;\ k'[\eta_c[i], k_c[i]] \leftarrow k;\ ent[\mu, k] \leftarrow ent[\mu, k] + 1\}$;
    **end**; % “if $k \le n$”
**END**
```

Now we are able to form the complete information specifying graph $G(\gamma) = (X, U)$. The set X of its nodes coincides with the set $\{O_{\eta,k} \mid k \in [n], \eta \in [\mu_k]\}$ of o-os. The set U of arcs is the union of two sets: $U = U' \cup U''$, where $U' \doteq \{((\eta, k), (\eta'[\eta,k], k'[\eta,k])) \mid (\eta,k) \in X'\}$ is the set of machine-based arcs, $U'' \doteq \{((\eta,k), (\eta + 1,k)) \mid k \in [n], \eta \in [\mu_k - 1]\}$ is the set of job-based arcs, and $X' \doteq \{(\eta, k) \mid k \in [n], \eta \in [\mu_k], k'[\eta,k] > 0\}$ is the subset of such o-os $O_{\eta,k} \in X$ for which there are subsequent o-os processed in schedule $S(\widehat{T}_\gamma)$ on the same machine $M_{\hat{i}[\eta,k]}$.

Next, after scanning (in any order) the set U of arcs of graph $G(\gamma)$, we can find for each node $O_{\eta,k} \in \mathcal{O}$ the list $U_{\eta,k}$ of outgoing arcs and the number $ent[\eta, k]$ of entering arcs. This information is sufficient to start the classical algorithm that in $O(|U|) \le O(|\mathcal{O}|) \le O(\mu n)$ time computes the active schedule $S_{\rm act}(\gamma)$ which meets precedence constraints specified on the set of o-os by graph $G(\gamma)$. Thus, Stage 7 can be performed in time $O(\mu\widetilde{N})$ needed for scanning all e-cells of table $\widehat{T}_\gamma$. This completes the description of Stage 7 and of the whole algorithm $\mathcal{A}(\gamma)$. ■

4. Analysis of algorithm $\mathcal{A}(\gamma)$ and formulation of the main result

Lemma 2. *For any $\gamma \in \mathbb{N}$, schedule $S_{\rm act}(\gamma)$ is feasible. If for some $\gamma \in \mathbb{N}$ schedule $S(\widehat{T}_\gamma)$ is also feasible, then $C_{\max}(S_{\rm act}(\gamma)) \le C_{\max}(S(\widehat{T}_\gamma))$.*

Proof. The feasibility of schedule $S_{\rm act}(\gamma)$ consists of the feasibility of sub-schedules for each machine and each job. For **each job**, this schedule is feasible, because graph $G(\gamma)$ specifies the linear order of execution of its operations based on the input information of the Job Shop problem. At the same time, the linear order of execution of the original operations of **each machine** M_i in graph $G(\gamma)$ is inherited from the linear order of execution of these operations in table-based schedule $S(\widehat{T}_\gamma)$, which ultimately ensures the feasibility of the active schedule $S_{\rm act}(\gamma)$.

Let us prove the relation $C_{\max}(S_{\rm act}(\gamma)) \le C_{\max}(S(\widehat{T}_\gamma))$ under the assumption that schedule $S(\widehat{T}_\gamma)$ is feasible. As was observed above, schedule $S(\widehat{T}_\gamma)$ meets the linear precedence constraints specified by graph $G(\gamma)$ for the set of all original operations on each machine M_i. Furthermore, due to the feasibility of schedule $S(\widehat{T}_\gamma)$, it meets the linear precedence constraints imposed on the set of all operations of each original job. Thus, $S(\widehat{T}_\gamma)$ meets all precedence constraints specified by graph $G(\gamma)$. Furthermore, schedule $S(\widehat{T}_\gamma)$ meets all precedence constraints specified by table $\widehat{T}(\gamma^*)$ on the set of **all operations** (including extended and S-dummy ones) of each machine M_i. Let $\widehat{G}(\widehat{T}_\gamma)$ be the graph with the set of nodes representing all operations of schedule $S(\widehat{T}_\gamma)$, and with the set of arcs including all transitive closures of precedence constraints specified by table $\widehat{T}_\gamma$ and graph $G(\gamma)$. Then, due to the inclusion $G(\gamma) \subseteq \widehat{G}(\widehat{T}_\gamma)$, the critical path in graph $\widehat{G}(\widehat{T}_\gamma)$ cannot be shorter than that of graph $G(\gamma)$, which implies the relation $C_{\max}(S_{\text{act}}(\gamma)) \le C_{\max}(S(\widehat{T}_\gamma))$. $\square$

Lemma 3. *For any $\gamma \ge \gamma^* \doteq \left\lceil \alpha(d)\mu\widetilde{N}\frac{p_{\max}}{L_{\max}}\right\rceil + 1$, schedule $S(\widehat{T}_\gamma)$ is feasible. For any instance of the Job Shop problem with $m \ge 2$ and $\mu \ge 2$, the length of schedule $S(\widehat{T}_{\gamma^*})$ meets the bound:*

$$C_{\max}(S(\widehat{T}_{\gamma^*})) < L_{\max} + m\mu^2(\mu-1)p_{\max}. \tag{15}$$

Proof. Let us first prove that the $\hat{k}$-property holds for table $\widehat{T}_\gamma$ with any $\gamma \in \mathbb{N}$ and for any $i \in [m]$. More precisely, we are to prove that the equalities

$$\hat{c}^{\,i}_{\eta,k}(\widehat{T}_\gamma) = \frac{k-d}{\widetilde{N}}L_{\max} + \frac{1}{\widetilde{N}}\sum_{\nu=1}^{\eta}\tilde{L}^i_\nu + \alpha(d)\sum_{\nu=1}^{\mu}\hat{\delta}^{\,i}_{\nu,k'(k,\nu,\eta)}(\gamma) \tag{16}$$

hold for all $i \in [m],\ \eta \in [\mu],\ k \in [\widehat{N}(\gamma)]$ for some values $\hat{\delta}^{\,i}_{\nu,k'(k,\nu,\eta)}(\gamma) \in [0, p_{\max}]$.

Indeed, as a result of adding S-dummy operations (each of length $\tilde{L}^i_\nu/\widetilde{N}$ for machine M_i) to the beginning and to the end of each row ν of table $\widetilde{T}$, formulas (11) transform into the equalities

$$\sum_{j\in[k]}\hat{p}^{\,i}_{\nu,j}(\widehat{T}_\gamma) = \frac{k-d+1}{\widetilde{N}}\tilde{L}^i_\nu + \alpha(d)\,\hat{\delta}^{\,i}_{\nu,k}(\gamma),\ \forall\ i \in [m],\ \nu \in [\mu],\ k \in [\widehat{N}(\gamma)], \tag{17}$$

with

$$\hat{\delta}^{\,i}_{\nu,k}(\gamma) \doteq \begin{cases} \delta^i_{\nu,k-\gamma(\nu-1)}, & \text{for } k-\gamma(\nu-1) \in [\widetilde{N}] \\ \frac{(d-1)}{\alpha\,(d)}\tilde{L}^i_\nu/\widetilde{N}, & \text{for } k \le \gamma(\nu-1) \text{ and } k > \gamma(\nu-1)+\widetilde{N}. \end{cases} \tag{18}$$

The second case of equalities (18) follows from relations $\sum_{j\in[k]}\hat{p}^{\,i}_{\nu,j}(\widehat{T}_\gamma) = k\frac{\tilde{L}^i_\nu}{\widetilde{N}}$ which hold for $k \le \gamma(\nu-1)$ and $k \ge \gamma(\nu-1)+\widetilde{N}$. Thus, in all cases, $\hat{\delta}^{\,i}_{\nu,k}(\gamma) \in [0, p_{\max}]$, because in the first case of (18) this is true for values $\delta^i_{\nu,j}$ $(j \in [\widetilde{N}])$, while in the second case this follows from $(d-1) < \alpha(d)$ and $\tilde{L}^i_\nu \le np_{\max} \le \widetilde{N}p_{\max}$.

Starting from (13) (with $T' = \widehat{T}_\gamma$) and implementing (17) and (9), we obtain for all $i \in [m],\ \eta \in [\mu],\ k \in [\widehat{N}(\gamma)]$:

$$\begin{aligned} \hat{c}^{\,i}_{\eta,k}(\widehat{T}_\gamma) &= \frac{k-d+1}{\widetilde{N}}\sum_{\nu=1}^{\eta}\tilde{L}^i_\nu + \frac{k-d}{\widetilde{N}}\sum_{\nu=\eta+1}^{\mu}\tilde{L}^i_\nu + \alpha(d)\sum_{\nu=1}^{\mu}\hat{\delta}^{\,i}_{\nu,k'(k,\nu,\eta)}(\gamma) \\ &= \frac{k-d}{\widetilde{N}}L_{\max} + \frac{1}{\widetilde{N}}\sum_{\nu=1}^{\eta}\tilde{L}^i_\nu + \alpha(d)\sum_{\nu=1}^{\mu}\hat{\delta}^{\,i}_{\nu,k'(k,\nu,\eta)}(\gamma), \end{aligned}$$

as required in (16). Now, to prove the feasibility of schedule $S(\widehat{T}_\gamma)$ for any $\gamma \ge \gamma^*$, we should prove that all precedence constraints (8) are met. To that end, it suffices to ensure the relations $0 \le s^{i''}_{\eta+1,k} - c^{i'}_{\eta,k}$ for all $i', i'' \in [m],\ \eta \in [\mu-1],\ k \in [\widetilde{N}]$ (i.e., for all extended jobs). As was observed at Stage 6 of the algorithm, operation $\widetilde{O}^{i''}_{\eta+1,k}$ coincides with operation $\widehat{O}^{i''}_{\eta+1,k+\gamma\eta}(\widehat{T}_\gamma)$, and so, $s^{i''}_{\eta+1,k} = \hat{s}^{\,i''}_{\eta+1,k+\gamma\eta}(\widehat{T}_\gamma) =$

$\hat{c}^{i''}_{\eta,k+\gamma\eta}(\widehat{T}_\gamma)$. Similarly, we obtain: $c^{i'}_{\eta,k} = \hat{c}^{i'}_{\eta,k+\gamma(\eta-1)}(\widehat{T}_\gamma)$. This together with (16) yields the following sufficient conditions for the feasibility of schedule $S(\widehat{T}_\gamma)$:

$$\begin{aligned} 0 \quad &\le \quad s^{i''}_{\eta+1,k} - c^{i'}_{\eta,k} = \hat{c}^{i''}_{\eta,k+\gamma\eta}(\widehat{T}_\gamma) - \hat{c}^{i'}_{\eta,k+\gamma(\eta-1)}(\widehat{T}_\gamma) = \frac{\gamma}{\widetilde{N}} L_{\max} + \frac{1}{\widetilde{N}}\sum_{\nu=1}^{\eta} \tilde{L}^{i''}_\nu \\ &+ \quad \alpha(d)\sum_{\nu=1}^{\mu} \hat{\delta}^{i''}_{\nu,k'(k+\gamma\eta,\nu,\eta)}(\gamma) - \alpha(d)\sum_{\nu=1}^{\mu} \hat{\delta}^{i'}_{\nu,k'(k+\gamma(\eta-1),\nu,\eta)}(\gamma) - \frac{1}{\widetilde{N}}\sum_{\nu=1}^{\eta} \tilde{L}^{i'}_\nu. \end{aligned}$$

To ensure all these inequalities, it suffices to take γ such that

$$\frac{\gamma}{\widetilde{N}} L_{\max} \ge \alpha(d)\sum_{\nu=1}^{\mu} \hat{\delta}^{i'}_{\nu,k'(k+\gamma(\eta-1),\nu,\eta)}(\gamma) + \frac{1}{\widetilde{N}}\sum_{\nu=1}^{\eta} \tilde{L}^{i'}_\nu$$

holds for any $i' \in [m],\ \eta \in [\mu-1],\ k \in [\widetilde{N}]$. E.g., $S(\widehat{T}_\gamma)$ is feasible for any

$$\gamma \ge \gamma^* \doteq \min\{\gamma \in \mathbb{N} \mid \gamma L_{\max}/\widetilde{N} \ge \alpha(d)\mu p_{\max} + L_{\max}/\widetilde{N}\} = \left\lceil \alpha(d)\mu\widetilde{N}\frac{p_{\max}}{L_{\max}} \right\rceil + 1.$$

Lastly, once schedule $S_i(\widehat{T}_{\gamma^*})$ (for any $i \in [m]$) has no idle time, its length is equal to $L_{\max}$ plus the total length of all S-dummy operations on machine M_i. Since each machine M_i has $\gamma^*(\mu-1)$ S-dummy operations at each level $\eta \in [\mu]$ (each of length $\tilde{L}^i_\eta/\widetilde{N}$), their total length over all levels is equal to $\gamma^*(\mu-1)L_{\max}/\widetilde{N}$, and we can estimate the length of schedule $S(\widehat{T}_{\gamma^*})$ as:

$$\begin{aligned} (19)\quad C_{\max}(S(\widehat{T}_{\gamma^*})) \quad &= \quad L_{\max} + \gamma^*(\mu-1)\frac{L_{\max}}{\widetilde{N}} < L_{\max} + \left(\alpha(d)\mu\widetilde{N}\frac{p_{\max}}{L_{\max}} + 2\right)\cdot \\ \cdot(\mu-1)\frac{L_{\max}}{\widetilde{N}} \quad &= \quad \left(1 + \frac{2\mu-2}{\widetilde{N}}\right) L_{\max} + \mu(\mu-1)(m\mu - 1 + 1/m\mu)p_{\max} \end{aligned}$$

Now, choosing the value $\widetilde{N} \leftarrow 3n$, we can estimate $\frac{2\mu-2}{\widetilde{N}} L_{\max} \le \frac{2(\mu-1)}{3n}\mu n p_{\max}$, which yields the bound:
$C_{\max}(S(\widehat{T}_{\gamma^*})) \le L_{\max} + \left(m\mu^2(\mu-1) + \left(\frac{2}{3}\mu(\mu-1) - \mu(\mu-1) + \frac{\mu-1}{m}\right)\right) p_{\max}$.

Since $m \ge 2$ and $\mu \ge 2$, the summand $\left(\frac{\mu-1}{m} - \frac{\mu(\mu-1)}{3}\right)$ in the above bound is always negative, which enables us to derive bound (15). Lemma 3 is proved. □

Lemma 4. *For any $\gamma \ge \widetilde{N}$, schedule $S(\widehat{T}_\gamma)$ is feasible. For any instance of the Job Shop problem, schedule $S(\widehat{T}_{\widetilde{N}})$ has length $C_{\max}(S(\widehat{T}_{\widetilde{N}})) = \mu L_{\max}$.*

Proof. Let $SO_i(\gamma)$ denote the sequence of operations executed on machine M_i according to schedule $S(\widehat{T}_\gamma)$. Then by construction of a table-based schedule (see its definition on page 14) and by the choice of $\gamma \ge \widetilde{N}$, the initial segment of $SO_i(\gamma)$ consisting of $\gamma\mu$ operations contains all $\widetilde{N}$ extended operations of the first level on that machine (and **only such** extended operations), $\gamma - \widetilde{N}$ S-dummy operations of the first level, and γ S-dummy operations of each level from 2 to μ. Thus, the total duration of this initial segment of operations on each machine M_i is equal to

$$\tilde{L}^i_1 + (\gamma - \widetilde{N})\frac{\tilde{L}^i_1}{\widetilde{N}} + \sum_{\eta=2}^{\mu} \frac{\gamma\tilde{L}^i_\eta}{\widetilde{N}} = \sum_{\eta=1}^{\mu} \frac{\gamma\tilde{L}^i_\eta}{\widetilde{N}} = \frac{\gamma L_{\max}}{\widetilde{N}}.$$

Similarly, we ensure that the next segment of $SO_i(\gamma)$ $(i \in [m])$ consisting of the same $(\gamma\mu)$ number of operations occupies in schedule $S(\widehat{T}_\gamma)$ a time interval of the same length ($\gamma L_{\max}/\widetilde{N}$) and contains all $\widetilde{N}$ extended operations (of machine M_i) of level 2, and so on, until the μ-th segment. This last segment is shorter: it contains only $\mu\widetilde{N}$ operations, with total length $L_{\max}$; still, it contains **all $\widetilde{N}$ extended operations** of the μ-th level on that machine. Thus, all operations of the η-th level on each machine M_i are concentrated in the η-th segment of the sequence of operations on that machine, while the η-th segments of all machines are synchronized in time. This provides the property that all extended operations of the $(\eta+1)$-th level are processed in this schedule after all operations of the η-th level, which provides the feasibility of schedule $S(\widehat{T}_\gamma)$.

The length of schedule $S(\widehat{T}_{\widetilde{N}})$ coincides with the total duration of μ segments of sequence $SO_i(\gamma)$ on each machine M_i, while the duration of each segment (in the case of $\gamma = \widetilde{N}$) is equal to $L_{\max}$. So, we have $C_{\max}(S(\widehat{T}_{\widetilde{N}})) = \mu L_{\max}$. □

Theorem 2. *Algorithm $\mathcal{A}(\gamma)$ with $\gamma = \gamma^{\#} \doteq \min\{\gamma^*, \widetilde{N}\}$ runs in time $O(m^2\mu^2 n^2)$ and finds a feasible schedule $S_{\text{act}}(\gamma^{\#})$ of length*

$$C_{\max}(S_{\text{act}}(\gamma^{\#})) \le \min\{mL_{\max}, \mu L_{\max}, L_{\max} + m\mu^2(\mu-1)p_{\max}\}. \tag{20}$$

Proof. Since in the case with either $m = 1$ or $\mu = 1$ the Job Shop problem can be solved to the optimum in $O(|\mathcal{O}|)$ time, we can further assume that the conditions $m \ge 2$ and $\mu \ge 2$ of Lemma 3 hold. This implies that algorithm $\mathcal{A}(\gamma^*)$ finds a feasible schedule $S(\widehat{T}_{\gamma^*})$ that meets bound (15). Next, we have relations

$$C_{\max}(S_{\text{act}}(\gamma^{\#})) \le C_{\max}(S(\widehat{T}_{\gamma^{\#}})) \le \min\{\mu L_{\max}, L_{\max} + m\mu^2(\mu-1)p_{\max}\}, \tag{21}$$

where the first inequality is valid by Lemma 2 (since schedule $S(\widehat{T}(\gamma^{\#}))$ is feasible in both cases of $\gamma^{\#} \in \{\widetilde{N}, \gamma^*\}$ due to Lemmas 4 and 3). The second inequality is valid due to the monotonic increase of the function $C_{\max}(S(\widehat{T}_\gamma))$ on γ and due to Lemmas 4 and 3. Lastly, bound $C_{\max}(S_{\text{act}}(\gamma^{\#})) \le mL_{\max}$ is valid, because schedule $S_{\text{act}}(\gamma^{\#})$ is feasible (by Lemma 2) and active. Along with (21), this yields bound (20).

To complete the proof of the theorem, it remains to justify the bound on the running time claimed above. Clearly, Stages 1 to 3 of algorithm $\mathcal{A}(\gamma)$ can be performed in time $O(m\mu n)$. The algorithm of vector summation performed at Stage 4 for the extended set of $\widetilde{N} = 3n$ jobs requires time $O(m^2\mu^2\widetilde{N}^2) \le O(m^2\mu^2 n^2)$. Next, as we could see from the description of Stage 7, neither table $\widetilde{T}$, nor table $\widehat{T}(\gamma^{\#})$ defined at Stages 5 and 6 were used for constructing the active schedule $S_{\text{act}}(\gamma^{\#})$, which means that Stages 5 and 6 of algorithm $\mathcal{A}(\gamma)$ are “imaginary” parts of it — they contribute nothing to the running time of the algorithm. (However, the structure of the table-based schedule $S(\widehat{T}(\gamma^{\#}))$ was essentially used while deriving our upper bound on the length of schedule $S_{\text{act}}(\gamma^{\#})$.) Finally, Stage 7 (of finding the active schedule $S_{\text{act}}(\gamma^{\#})$) can be performed in time $O(\mu\widetilde{N}) \le O(\mu n)$. Totally, this all takes time $O(m^2\mu^2 n^2)$. □

5. Linearization of algorithms from (Shmoys et al., 1994) with respect to the number of jobs (n)

Two algorithms published in (Shmoys et al., 1994) were briefly described in the Introduction (with mentioning the role that our algorithm from (Sevast’yanov, 1986) plays in both of them, see pages 5–6).

In both those algorithms, the set of all jobs $\mathcal{J}$ is first separated into two subsets ($\mathcal{J}_L$ and $\mathcal{J}_S$) of so-called "large" and "small" jobs, after which **different algorithms** are applied to them, constructing two schedules independent of each other. (In particular, to construct a schedule for "small" jobs, our algorithm from (Sevast'yanov, 1986), based on the compact vector summation technique, was used.) The "watershed" between two sets of jobs is established at the input of the *Algorithm of job separation* by specifying a *threshold value* $p^{\#}$ of the parameter $\beta(p_j)$ (where p_j is the vector of operation durations of job J_j, and $\beta(x)$ is the maximum component of vector x): jobs with $\beta(p_j) \leq p^{\#}$ fall into $\mathcal{J}_S$, the rest ones — into $\mathcal{J}_L$. Thus, the value of $p_{\max}$ over all small jobs is not greater than the value of the "watershed" $p^{\#}$.

This value is taken differently in the two algorithms: $p^{\#} = \frac{L_{\max}}{2m\mu^3}$ in the first algorithm and $p^{\#} = \frac{\varepsilon L_{\max}}{2m\mu^3}$ in the second one. As a result, the first algorithm from (Shmoys et al., 1994) provides a relative error bounded by $O(\log^2(m\mu))$, while the second algorithm yields a $(2+\varepsilon)$-approximation for any fixed $\varepsilon > 0$. At that, the bounds on the running times of both algorithms ($O(n^2m^2\mu^2 + P(m,\mu))$, where $P(m,\mu)$ is a polynomial of m and μ, and $O(n^2)$, under assumption that parameters m and μ are fixed, respectively) show that they heavily depend on the algorithm from (Sevast'yanov, 1986). We are now to show that the bounds on the running times of both these algorithms can be made linear in n.

Linearization of both algorithms from (Shmoys et al., 1994) in the parameter *number of jobs* can be achieved (after the separation of the set of jobs into "large" and "small") by a preliminary gluing together "small" jobs to "aggregated small jobs", the number of which (due to the choice of the threshold value $p^{\#}$) is bounded from above by an amount independent of the initial number of jobs (n). This value is a polynomial in m and μ in the case of the first algorithm from (Shmoys et al., 1994) and a polynomial in m, μ, and $1/\varepsilon$ in the case of their second algorithm. Since the "aggregated small jobs" also meet the threshold value $p^{\#}$, applying to them our algorithm from (Sevast'yanov, 1986) or algorithm $\mathcal{A}(\gamma)$ from Sect. 3, we obtain a feasible schedule for the aggregated (and hence for the original) small jobs, which has **exactly the same upper bound on its length** as we would have obtained for small jobs without gluing them together. The only difference is that the complexity of this algorithm will no longer depend on n, but will be a polynomial in m and μ (and of $1/\varepsilon$ in the case of the second algorithm). Thus, the dependence on parameter n is preserved only for the complexity of the gluing procedure itself, which can be performed in time $O(\sum_{j\in[n]} \mu_j) \leq O(\mu n)$ (i.e., in time needed to look through the input information) by means of the procedure *Job_ Separation_ Gluing_ Unification* (further — *JSGU*, for short) described below.

In this procedure, three different processes are combined in a natural way within **a single loop of scanning the list of original jobs**: (1) Stage 1 of algorithm $\mathcal{A}(\gamma)$ (where the unification of job routes through machines is performed), (2) job separation (into "large" and "small" ones), and (3) the gluing the small jobs.

Prior to the description of the procedure, we should make two remarks. First, we note that after the status of a job (being large or small) is defined, large jobs have not be subjected to processes of unification and gluing, as well as to the subsequent stages of algorithm $\mathcal{A}(\gamma)$, since other algorithms (not based on the compact vector summation) are used in (Shmoys et al., 1994) for constructing the schedules for large jobs. Thus, since the *JSGU*-procedure includes the job unification, it is intended to **replace Stage 1 of algorithm** $\mathcal{A}(\gamma)$ applied to small jobs.

Second, it can be observed that the procedure of gluing the small jobs and the unification of their routes can be performed simultaneously. Indeed, at Stage 1 we define *wide jobs* ($\{\tilde{J}_j\}$), each consisting of $m\mu$ "wide operations" $\{O^i_{\eta,j} \mid i \in [m],\ \eta \in [\mu]\}$. The duration $p^i_{\eta,j}$ of each such operation is first set to zero. Then, for each job $J_j \in \mathcal{J}$, we plunge its o-os $\{O_{\eta,j} \mid \eta \in [\mu_j]\}$ into "wide operations" by setting $p^{\hat{\imath}(\eta,j)}_{\eta,j} \leftarrow p^{\hat{\imath}(\eta,j)}_{\eta,j} + p_{\eta,j}$, where $M_{\hat{\imath}(\eta,j)} = M(O_{\eta,j})$. But in fact, **several original small jobs** J_j (all together) could be loaded into **the same wide job** $\tilde{J}_k$ so as the total duration $\check{p}^i_{\eta,k}$ of their operations within each *wide operation* $O^i_{\eta,k}$ would not exceed the given threshold $p^{\#}$. (This process can be viewed as a packing of vectors $\{p_j\}$ into an $(m\mu)$-dimensional bin $\tilde{J}_k$.) So, the jobs placed into the same wide job $\tilde{J}_k$ could be glued together, thereby forming an "aggregated small" wide job.

We next observe that the process of packing the original jobs into a **fixed wide job** $\tilde{J}_k$ can be continued until the attempt to place the next in turn vector $p_{j'}$ **of a small job** $J_{j'}$ into the current bin $\tilde{J}_k$ yields an overload of some cell of that $(m\mu)$-dimensional bin. In this case, we proceed quite simply: we treat bin $\tilde{J}_k$ as "filled" and add the "aggregated small" wide job $\tilde{J}_k$ (with a total vector $\check{p}_k \in \mathbb{R}^{m\mu}$ of operation durations) to set $\mathcal{J}_S$. After that, we start filling the next bin ($\tilde{J}_{k+1}$) by placing vector $p_{j'}$ into it. Finally, if job $J_{j'}$ is "large", we add it to set $\mathcal{J}_L$. (These jobs will not be further involved in our algorithm.)

Thus, our procedure (combining the three processes mentioned above) represents a loop of scanning the list $\mathcal{J}$ of the original jobs (in any order), where, while scanning some job $J_j \in \mathcal{J}$, we clarify two questions:

(A) Does the job fit into the current bin $\tilde{J}_k$? (Does $\check{p}^{\hat{\imath}(\eta,j)}_{\eta,k} + p_{\eta,j} \le p^{\#}$ hold for all $\eta \in [\mu_j]$?)

(B) Is it "small" or "large"? (Does $p_{\eta,j} \le p^{\#}$ hold for all $\eta \in [\mu_j]$?)

Clearly, $2\mu_j$ comparisons are sufficient to answer both questions. Yet, this can be done faster: by performing at most $(\mu_j + 1)$ (at best, two) comparisons. This can be done as follows (see the description of the procedure below for details).

So, while considering some job $J_j \in \mathcal{J}$, we start the loop on its operations $\{O_{\eta,j} \mid \eta = 1, 2, \ldots, \mu_j\}$. For each value of η, we first try to answer only the first question: does operation $O_{\eta,j}$ fit into the current "aggregated wide operation" $O^{\hat{\imath}(\eta,j)}_{\eta,k}$? (If it fits, then it is small.) If for all values of η we have the positive answer to this question, we get positive answers to both questions ((A) and (B)) on job J_j: it is small and it fits into the bin.

In case that a value $\eta = \eta^*$ is encountered such that $\check{p}^{\hat{\imath}(\eta^*,j)}_{\eta^*,k} + p_{\eta^*,j} > p^{\#}$ (and so, we have got the negative answer to question (A), while it still remains unclear, if job J_j is small or large), we start (for values $\eta = \eta^*$ and on) getting answers to question (B): is operation $O_{\eta,j}$ small or large? (Or: does $p_{\eta,j} \le p^{\#}$ hold?) If for all η the answers are positive, job J_j is small. Otherwise, it is large. (It should be included in list $\mathcal{J}_L$ and taken out of the operation of our algorithm.)

Procedure *JSGU* (described below) uses the following *global variables*:

integer K is the number of "aggregated small" wide jobs;

$\check{p}_k$ ($k \in [K]$) is an $(m\mu)$**-dimensional vector of integers** accumulating the resulting operation durations of the k-th "aggregated small" wide job;

$\mathcal{I}_L$ and $\mathcal{I}_S(k)$ ($k \in [K]$) are **sets of integers** accumulating the sets of the original indices of large jobs and of small jobs plunged into the k-th aggregated small wide job, respectively. All other information on the original jobs is also global.

```
                    **Procedure** $JSGU$(**integer:** $n, p^{\#}$);
% **Formal input parameters:**
% $n$ is the number of original jobs in $\mathcal{J}$;
% $p^{\#}$ is the *upper bound* on the maximum operation duration for “small” jobs

% **Local parameters:**
  **integer:** $i, j, k, \eta, D$;
  **Boolean**: **unfit**, **large**;

  **Procedure** $Load$(**integer:** $j, k$); % load job $j$ into bin $k$
  **integer:** $\eta$;
  **begin**
    $\mathcal{I}_S(k) \leftarrow \mathcal{I}_S(k) \cup \{j\}$;
    **for** $\eta \leftarrow 1, \ldots, \mu_j$ **do** $\check{p}_{\eta,k}^{\hat{i}(\eta,j)} \leftarrow \check{p}_{\eta,k}^{\hat{i}(\eta,j)} + p_{\eta,j}$
  **end** % Procedure $Load$

  **BEGIN**
  $\mathcal{I}_L \leftarrow \varnothing$; $k \leftarrow 1$; $\mathcal{I}_S(k) \leftarrow \varnothing$; $\check{p}_k \leftarrow \mathbf{0}$;
  **for** $j \leftarrow 1, \ldots, n$ **do begin**
    **unfit** $\leftarrow$ FALSE; **large** $\leftarrow$ FALSE; $\eta \leftarrow 1$;
    % Initially we assume that job $J_j$ is small and fits into the current bin.
    % Next, we run the loop on its operations (indexed by $\eta$):
    **repeat if** $(\check{p}_{\eta,k}^{\hat{i}(\eta,j)} + p_{\eta,j} \leq p^{\#})$ **then** $\eta \leftarrow \eta + 1$ **else unfit** $\leftarrow$ TRUE
    **until** $(\eta > \mu_j) \vee$ **unfit**;
    **if unfit then repeat if** $(p_{\eta,j} \leq p^{\#})$ **then** $\eta \leftarrow \eta + 1$ **else** (**large** $\leftarrow$ TRUE)
    **until** $(\eta > \mu_j) \vee$ **large**;
    **if not unfit** % which means that job $J_j$ is small and fits into the current bin
    **then** $Load(j, k)$
      **else if not large then** $\{k \leftarrow k + 1;\ \mathcal{I}_S(k) \leftarrow \varnothing;\ \check{p}_k \leftarrow \mathbf{0};\ Load(j, k)\}$
        **else** $\mathcal{I}_L \leftarrow \mathcal{I}_L \cup \{j\}$;
  **end** % for $j$
  $K \leftarrow k$;
  **if** $k > 2$ **then begin**
    $D \leftarrow 0$; **for** $i \leftarrow 1, \ldots, m$ **do for** $\eta \leftarrow 1, \ldots, \mu$ **do** $D \leftarrow D + \check{p}_{\eta,1}^{i} + \check{p}_{\eta,k}^{i}$;
    **if** $D \leq p^{\#}$ **then** $\{\check{p}_1 \leftarrow \check{p}_1 + \check{p}_k;\ \mathcal{I}_S(1) \leftarrow \mathcal{I}_S(1) \cup \mathcal{I}_S(k);\ K \leftarrow k - 1\}$
    % If the total length ($D$) of aggregated small jobs $\tilde{J}_1$ and $\tilde{J}_k$ does not exceed
    % $p^{\#}$, we combine them; if $D > p^{\#}$, we do nothing (thus retaining the value
    % $K = k$; in particular, $D > p^{\#}$ is guaranteed in case $k = 2$).
  **end** % if $k > 2$
  **END**;
```

Lemma 5. *Procedure $JSGU(n, p^{\#})$ runs in time $O(\mu n)$. The resulting number K of aggregated small jobs and the number n_L of large jobs meet the relation*

$$K/2 + n_L < mL_{\max}/p^{\#}. \tag{22}$$

Proof. The procedure requires a single scanning of the set of original jobs, while the information on each job $j \in [n]$ is bounded by $O(\mu_j)$. This yields the bound on the running time of the procedure.

Let $d(x)$ denote the sum of components of vector x, $\langle \check{p}'_\ell, K' \rangle$ and $\langle \check{p}''_\ell, K'' \rangle$ denote the values of vector $\check{p}_\ell$ (of operation durations of the aggregated small wide job $\tilde{J}_\ell$) and of parameter K (the resulting number of aggregated small wide jobs) right before and after performing the "if $k > 2$" block, respectively. According to the *JSGU*-procedure, at least one component of each vector $\check{p}'_\ell + \check{p}'_{\ell+1}$ ($\ell \in [k-1]$) is greater than $p^{\#}$ (while other components are non-negative), which implies

$$d(\check{p}'_\ell) + d(\check{p}'_{\ell+1}) = d(\check{p}'_\ell + \check{p}'_{\ell+1}) > p^{\#}, \ \forall\, \ell \in [k-1]. \tag{23}$$

After performing the block "if $k > 2$", we have one of two cases: either (a) $K'' = k$, or (b) $K'' = k - 1$ (when we combine the aggregated jobs $\tilde{J}_1$ and $\tilde{J}_k$, and vector $\check{p}_1$ receives the value $\check{p}''_1 \doteq \check{p}'_1 + \check{p}'_k$). Thus, in case (b), we have:

$$d(\check{p}''_1) + d(\check{p}''_{K''}) = d(\check{p}'_1 + \check{p}'_k) + d(\check{p}'_{k-1}) = d(\check{p}'_1) + d(\check{p}'_{k-1} + \check{p}'_k) > d(\check{p}'_1) + p^{\#}, \tag{24}$$

while (23) in terms of vectors $\{\check{p}''_\ell\}$ transforms into

$$d(\check{p}''_1) + d(\check{p}''_2) = d(\check{p}'_1 + \check{p}'_k + \check{p}'_2) = d(\check{p}'_1 + \check{p}'_2) + d(\check{p}'_k) > p^{\#} + d(\check{p}'_k); \tag{25}$$

$$d(\check{p}''_\ell) + d(\check{p}''_{\ell+1}) > p^{\#}, \ \ell \in \{2, \ldots, k-1\}. \tag{26}$$

Summing up inequalities (24)–(26), we obtain:

$$2 \sum_{\ell \in [K'']} d(\check{p}''_\ell) > K'' p^{\#} + d(\check{p}'_1 + \check{p}'_k).$$

Since in case (a), we have $K'' = K' = k$ and $\check{p}''_\ell = \check{p}'_\ell, \ \forall\, \ell$, we obtain:

$$d(\check{p}''_1) + d(\check{p}''_{K''}) = d(\check{p}'_1) + d(\check{p}'_k) > p^{\#}, \text{ and, using (23):}$$
$$d(\check{p}''_\ell) + d(\check{p}''_{\ell+1}) = d(\check{p}'_\ell) + d(\check{p}'_{\ell+1}) > p^{\#}, \ \forall\, \ell \in [K''-1].$$

Similarly summing up these inequalities, we obtain the inequality:

$$2 \sum_{\ell \in [K'']} d(\check{p}''_\ell) > K'' p^{\#}, \tag{27}$$

which holds in both cases ((a) and (b)). Similarly, for "large" job, we have:

$$\sum_{j \in \mathcal{I}_L} d(p_j) > n_L p^{\#}. \tag{28}$$

Using these two bounds on the total durations of "aggregated small" and of "large" jobs, and recalling that their total duration is at most $mL_{\max}$, we obtain:

$$mL_{\max} \geq \sum_{\ell \in [K'']} d(\check{p}_\ell) + \sum_{j \in \mathcal{I}_L} d(p_j) > \frac{K''}{2} p^{\#} + n_L p^{\#},$$

which implies (22). Lemma 5 is proved. □

Corollary 2. *Executing the $JSGU(n, p^{\#})$ procedure instead of Stage 1 of algorithm $\mathcal{A}(\gamma)$ applied for constructing a schedule for "aggregated small" jobs in the first and second algorithms from (Shmoys et al., 1994) (with values $\frac{L_{\max}}{m\mu^2(\mu-1)}$ and $\frac{\varepsilon L_{\max}}{m\mu^2(\mu-1)}$ of parameter $p^{\#}$, respectively) provides the number n_S of "small" jobs which meets upper bounds: $n_S < 2m^2\mu^2(\mu-1)$ and $n_S < 2m^2\mu^2(\mu-1)/\varepsilon$, respectively. This provides an approximate solution for the original instance of the Job Shop problem* *with a running time esimated as $O(\mu n+f(m,\mu))$ and $O(\mu n+g(m,\mu,\varepsilon))$, respectively, where $f(m,\mu)$ is a polynomial of m and μ, while function $g(m,\mu,\varepsilon)$ exponentially depends on parameters m,μ, and $1/\varepsilon$.*

6. Comparative analysis of four theoretical approximation algorithms for the Job Shop problem

When there are several alternative ways to solve a problem, the question of comparing these ways naturally arises. We want to find out in which cases we should give preference to one solution method and in which cases to another ones. For that purpose, it is desirable to have tools that enable one to conduct an objective analysis of the quality of methods and to compare the methods with each other.

As for "tools", in recent decades the methods of "experimental analysis" and "experimental comparison" of algorithms have become widespread. These methods are based on programming algorithms and on subsequent running of those programs on some **finite sets of instances** of the problem under consideration. The results of these calculations for different programs/algorithms are compared with each other, and from this (usually ambiguous) comparison a conclusion is drawn most suitable to the author of such a comparison. However, the results of this kind pertain to another science, called "experimental mathematics", and have no relation to the science (mathematics) to which this article is dedicated.

It seems that an objective comparison of algorithms is possible only in those cases when theoretically justified guarantees of their quality (such as accuracy bounds and bounds on their running time) have been obtained. However, such results are encountered in scheduling literature several orders rarer than those mentioned above. For example, for the Job Shop problem in its general form (i.e., for one of the most difficult-to-solve representatives of the area of *multistage scheduling problems*), **only four** (as far as I know, over the more than 70-year history of this field) such polynomial-time approximation algorithms with guaranteed a priori accuracy bounds valid **for all instances of this problem** have been developed. (Either detailed or brief descriptions of these four algorithms **were given** in the previous sections and in the Introduction.)

However, even the presence of such bounds does not always provide the opportunity for comparing different algorithms with each other, since those bounds may turn out to be incomparable. This may happen for different reasons. For example, they may be formulated in different terms that are not comparable with each other. In addition, a typical situation is when different methods may have their advantages in different sub-domains of the problem definition domain, but no method dominates absolutely in the entire domain. Finally, there may be no "absolute winner" for that simple reason that the problem of comparison is **multi-criterion**.

In our case, when we limited ourselves to evaluating the quality of algorithms by only two criteria: the accuracy of the solutions obtained and the complexity of the algorithm, the objective of a theoretical identification of a "winner" among the algorithms is difficult already due to the multi-criteria nature of the compared objects. Yet, even when comparing only by one criterion (for example, by complexity), certain difficulties may arise in those cases when the bounds on the running times of candidate algorithms depend on several independent parameters (as is in the case of the Job Shop problem). Alternatively, if we are talking about accuracy bounds (even without comparing algorithms by their complexity), then, in my opinion, the bounds on absolute and relative errors have **different physical dimensions**, and for this reason **are incomparable**. Comparing such bounds is the same as trying to answer the question: "Who jumps better: a long jumper or a high jumper?"

To tell the truth, it is not difficult to transform the absolute bound (15) into a relative one: it suffices to divide both parts of the inequality by $L_{\max}$ and recall that $L_{\max}$ is a lower bound on the optimum. Right such a relative bound

$$\frac{C_{\max}(S)}{C^*_{\max}} \le \frac{C_{\max}(S)}{L_{\max}} \le 1 + \frac{m\mu^3 p_{\max}}{L_{\max}} \tag{29}$$

implies the asymptotic optimality of our solutions under the condition that $n \to \infty$.

Ok! We now have *a priori* **relative accuracy bounds** for all four algorithms. At that, all of them are parametric and all of them estimate the **worst case** (for given values of the parameters in terms of which they are expressed). Let us try to compare them. And before starting this process, let us collect in one table the information we have about the four algorithms mentioned above, according to two criteria: (1) the accuracy of the solutions obtained and (2) the running time of the algorithm. We will number the algorithms in chronological order: **1** — (Sevast'yanov, 1986), **2** — (Shmoys et al., 1994) (1st), **3** — (Shmoys et al., 1994) (2nd), **4** — (Jansen et al., 2003).

Alg.	**Accuracy bound**	**Run-time**	**Parameters**	**Applicability**
1	(29)	$O(m^2\mu^2n^2)$	Any m, μ, n	Yes
2	$\frac{C_{\max}(S)}{OPT} \le O(\log^2(m\mu))$	$O(m^2\mu^2n^2)$	Any m, μ, n	Bad accuracy
3	$\frac{C_{\max}(S)}{OPT} \le 2+\varepsilon$	$O(n^2)$	Fixed m, μ, ε	No
4	$\frac{C_{\max}(S)}{OPT} \le 1+\varepsilon$	$O(n)$	Fixed m, μ, ε	No

Table 1. Comparison of the four algorithms

Can we start comparing these four algorithms now?

No, they still remain **incomparable in accuracy** (except for the two algorithms heaving bounds depending only on ε; they **can be compared**), since they depend on different sets of parameters. Namely, bound (5) depends on parameters m and μ, bound (6) by (Shmoys et al., 1994) and bound $(1+\varepsilon)$ for each algorithm $\mathcal{A}_\varepsilon$ by (Jansen et al., 2003) depend on ε, while our bound (29) depends on two extra parameters: $p_{\max}$ and $L_{\max}$, and does not depend on ε. However, these differences are easily leveled off by Shmoys et al.: they just put $p_{\max} = L_{\max}$ in our bound (29), which is a gross distortion of the bound. Of course, for some (very rare) instances (with small values of parameters m, μ and n) these parameters can take close values. But such an event is practically improbable for instances of practical size. For the **overwhelming majority** of practically significant instances, our accuracy bound (due to its asymptotic optimality) outperforms bound (6) (not mentioning (5)).

However (Shmoys et al., 1994), having performed the above transformation of my accuracy bound (i.e., having equated $p_{\max}$ to $L_{\max}$), came to a completely different conclusion: "It is interesting to note that Sevast'yanov's algorithm for the job shop problem can be viewed as a $(1 + m\mu^3)$-approximation algorithm". This "interesting observation" and a comparison of the accuracy bound obtained in this way with their own (obviously better!) bounds on the relative error enabled the authors to make an irrefutable conclusion that their algorithms are "better than previously known ones" (which is reflected in the title of their article: "Improved Approximation Algorithms...")[4]. It had not come to their mind that I would never publish an algorithm with that terrible relative accuracy bound, just because the simplest algorithm $\mathcal{A}_{02}$ constructs a $\min\{m,\mu\}$-approximate schedule in $O(n\mu + m\mu)$ time (see Lemma 1 on page 9).

As for the evaluation and comparison of algorithms by the second criterion (the running time), Shmoys et al. (1994) admit that their algorithms are inefficient, but they assign the responsibility for that inefficiency on my algorithm: "While all of the algorithms that we give are polynomial-time, they are all rather inefficient. Most rely on the algorithms of Sevast'yanov." However, I would look at this differently.

Of course, the authors of (Shmoys et al., 1994) are right that the complexity of their algorithms depends significantly on the complexity of our algorithm, for the simple reason that our algorithm is the **most significant part of their algorithms**, applied to **the majority of jobs** in the original problem instance (to so-called *small jobs*). For this reason, the bound on the running time of their **first algorithm** coincides with that of ours. Which, however, is not a "death sentence" for it, since the bound $O(m^2\mu^2n^2)$ **is completely polynomial in all parameters** of the problem, and is only quadratic in the main parameter, the number of jobs n, whereas the values of the other two parameters, m and μ, are usually relatively small, and, while solving the problem for a fixed shop, they "hit into" their natural upper limits, when the planning horizon is expanded. Taking into account the possibility of linearization of the complexity of this algorithm in n, demonstrated in Section 5, and also due to the high speed of modern Supercomputers (*SC*, for short), such complexity of the algorithm does not seem to be an insurmountable obstacle for its application to instances of practical size. However, this does not make this algorithm competitive with ours, since in accuracy it remains catastrophically worse than our (asymptotically optimal) algorithm **on the majority of instances of practically significant size**.

As for the second, $(2+\varepsilon)$-approximation algorithm from (Shmoys et al., 1994) (the complexity of which, after linearization, can be represented as $O(\mu n)+\psi(m,\mu,\varepsilon)$, where $\psi(m,\mu,\varepsilon)$ is an exponent of m,μ, and $1/\varepsilon$), its main complexity (and the fundamental inapplicability of the algorithm for solving practical problems) lies not in our algorithm, but **is hidden in the "constants"** that estimate the complexity of the **second half of the algorithm**. And as will be shown below, neither the linearization by n of the running time of the first half of this algorithm, nor any technical progress in the design of supercomputers can correct this situation.

To convince the reader in the practical inapplicability of this algorithm, we will suggest to it to find a $(2+\varepsilon)$-approximate solution for a definitely small-size instance with $m = 10$ machines in the shop and with $\mu = 10$ operations per job (all of which are to be processed on 10 **different machines**)[5]. At that, we will take the number

[4]In a similar way, i.e., ignoring other conditions and parameters of comparison and taking into account only the parameters chosen by themselves, they could conclude that "The long jumper jumps better!".

[5]This special case of the Job Shop problem is called an *acyclic job shop problem*, in which each job has exactly one operation on each machine, and each machine M_i processes n jobs according to some permutation $\pi^i = (\pi^i_1, \ldots, \pi^i_n)$.

of jobs $n \approx 10000$, which clearly does not exceed its average value for practically significant instances. We will neither overstrain their algorithm by proposing to find solutions arbitrarily close to 2-approximate ones, but take instead $\varepsilon = 8$ (thus letting the algorithm find just a 10-approximate solution for our small instance).

Finally, we will allow the authors to use our new accuracy bound (15) (which should help reduce the running time of their algorithm). We will also make five further agreements.

A. That our instance is being solved in a *System* of parallel Supercomputers, each SC having the maximal known for today capacity of $2 \cdot 10^{18}$ flops.

B. That there are 10 billion such SCs in our System on the Globe, i.e., about one "personal Supercomputer" for every inhabitant of the Earth[6].

C. That in each of 10 trillion galaxies (in the visible part of the Universe) there are 100 billion stars, and each star has 10 planets, on each of which we have placed an SC-system similar to that of the Earth. Totally, 10^{25} Globe SC-systems, forming the *Supercomputer System of the Universe* (*SSU*, for short).

D. That we have the opportunity to parallelize our problem, equally dividing the load between all computers of the SSU, and that each SC can work as long and continuously as desired.

E. That we measure the working time of our SSU in time units "*AU*" (*Age of the Universe*). We took the value of this amount to be **15** billion years.

After that much strong assumptions (and suggestions on our part), the reader should have absolutely no doubts about the "success of the enterprise". However, the results of the forthcoming analysis may be very discouraging to those potential users of this algorithm who were attracted by its theoretical performance guarantees in accuracy and complexity given in (Shmoys et al., 1994).

To begin with, let us estimate the total capacity of our SSU, as well as how many flops (floating point operations) it can perform per the new unit of time, 1AU.

Taking into account the above characteristics of the System, its total capacity is $2 \cdot 10^{18} \cdot 10^{10} \cdot 10^{25} = 2 \cdot 10^{53}$ flops. Accepting the agreement that the Earth year consists of 365.25 days, we obtain the relation: $1\text{AU} = 1.5 \cdot 10^{10} \cdot 365.25 \cdot 24 \cdot 3600 \approx 4.73 \cdot 10^{17}$ seconds. Multiplying this by the capacity of SSU, we obtain an upper bound on the capacity of SSU per 1AU: 10^{71} flopAU.

Now let us estimate the complexity of an approximate solution to our problem instance by the second algorithm from (Shmoys et al., 1994). The user of this algorithm will have to find an optimal schedule for the set of so called *large jobs*, the number of which can be estimated from above by $n' = m^2\mu^2(\mu - 1)/\varepsilon = 11250$. If we solve this sub-problem by a simple enumeration of all combinations of permutations $\{\pi^i \mid i \in [m]\}$ (I remind that this sub-problem must be solved **exactly**, so as to guarantee the relative error of 10 for the resulting solution), then this will require enumerating $(n')!^m \approx (11250/\mathbf{e})^{112500} \cdot (2\boldsymbol{\pi} \cdot 11250)^5 \approx 10^{112500 \cdot \log_{10} 4138.644} \cdot 10^{5 \log_{10} 70\ 650} \approx 10^{406\ 921}$ variants of combinations of job permutations. For each variant of this schedule, graph G of precedence constraints specifying the precedence of operations by jobs and by machines, contains $mn' = 112500$ nodes and about $2mn' = 225000$ arcs. It should be checked for the absence of directed cycles, and if there are no cycles, the length of the critical path in graph G should be computed. Using for these purposes the algorithm with running time $O(mn')$ (linear in the

[6]By the way, ONE such modern SC occupies an area greater than 1000 m^2, which results in total more than 10.000.000 km^2, — more than the area of China.

number of arcs of the graph), we have to multiply the above amount by at least the number $225000 \approx 10^{5.35}$ of arcs in G, which results in $10^{406\ 926}$ elementary actions to be performed. Thus, to perform these calculations on our CSU (the **Super-Computer System of the Univers**) and find a solution (to a given small instance) that would be no more than **10 times worse the optimum**, they would need time equal to $10^{406\ 855}$ **Ages of the Universe**.

As for the question of the practical feasibility of algorithms $\mathcal{A}_\varepsilon$ from the PTAS designed by Jansen et al. (2003), preliminary estimation shows that the situation for these algorithms is even more deplorable starting from rather small values ??of parameters m and μ. (At that, the scheme works only with values $\varepsilon < 1$.)

7. Conclusion

In this paper, we present a new version of the algorithm for solving the Job Shop problem based on the compact vector summation algorithm. As shown in the paper, the new algorithm guarantees an improved a priori bound on the absolute error of the solutions obtained, while the bound on the running time remains the same. In addition, in Section 6 we perform a comparative analysis of quality of four currently known theoretical algorithms of approximate solution of this problem with a priori performance guarantees. Undoubtedly, the algorithms presented in (Shmoys et al., 1994) and (Jansen et al., 2003) have significant theoretical value from the view point of **parametric analysis of the complexity of approximate solution** of the Job Shop problem. However, in Section 6 we pursued a different goal: to perform a comparative analysis of the **practical suitability** of these algorithms for solving real-size instances of the Job Shop problem. The results of this analysis and of the comparison of the four algorithms are the following conclusions (the first two relate to accuracy, the rest ones — to the complexity of the algorithms).

(1) According to the theoretical accuracy bounds derived for these algorithms, **none of them dominates** the others **in the entire range of instances** of this problem (in terms of accuracy of the solutions obtained).

(2) For the overwhelming majority of instances having practically significant sizes, our algorithm with bound (29) (ensuring **asymptotic optimality** of the solutions obtained with an increase in the planning horizon) has an advantage in accuracy over the algorithms from (Shmoys et al., 1994).

(3) Our algorithm has a moderate, polynomial in all main parameters, bound on the running time $O(m^2\mu^2n^2)$, which is just quadratic in the main parameter of the problem, the number of jobs n. At that, the other two parameters, m and μ, have natural upper bounds independent of length of the planning horizon (T), and thus, in the conditions of solving a practical problem **for a fixed shop**, with an increase in T they can be treated as amounts ??limited by constants, which justifies the **practical feasibility** of our algorithm on **real size problem instances** on modern computers.

(4) The first of the two algorithms from (Shmoys et al., 1994) in its original version has no advantage in complexity over our algorithm, but after its modification (linearization in n, performed in Section 5) it gains some advantage in the running time for large values of n. At that, as n increases, it becomes **much worse than our algorithm** in accuracy (due to the fact that its accuracy bound, depending on m and μ, remains stable, while the relative error of our algorithm tends to zero).

(5) The second algorithm from (Shmoys et al., 1994), supposedly guaranteeing $(2+\varepsilon)$-approximation in polynomial time (for any fixed values of parameters m, μ, and ε), is **practically inoperative** on the overwhelming majority of instances (starting with instances of small size), since even **after its linearization** in n it is not capable of obtaining even 10-approximate solutions for small enough instances **in any physically observable time**. Thus, **its comparison with our algorithm makes no sense** on the majority of real-size instances.

The situation with the practical applicability of any algorithm $\mathcal{A}_\varepsilon$ from the PTAS presented in (Jansen et al., 2003) (according to our preliminary estimation) seems even more hopeless. Moreover, as our calculations performed in Section 6 convincingly demonstrate, this deplorable situation with the last two algorithms can be corrected by no future progress in the development of Supercomputers.

The calculations made in Section 6 raise up some further interesting questions:

— What are the huge Super-Computers needed for, if, even with the help of algorithms with "improved" performance guarantees, the Super-Computer Army of the whole Universe for the whole Eternity is not capable of finding a rough approximate solution for a small instance of an ordinary (just strongly NP-hard) discrete optimization problem?

— What are the ingenious algorithms with theoretically justified "excellent performance guarantees" (high accuracy and "polynomial-time" complexity) needed for, if, even with the power of the Super-Computer System of the Universe and for the whole Age of the Universe they are unable to find a rough approximate solution for a small problem instance?

The answer to the first question, apparently, is that Super-Computers are dedicated to solving some other-type enumeration problems (consisting in processing a large amount of information in a minimum time and allowing maximum parallelization of this process). And even when they are used for solving such problems as the Job Shop problem, they should run programs for the algorithms having more definite performance guarantees (without hiding their inaccuracy into forms like "$O(\cdot)$" and hiding their running time into "constants" with indefinite values). When any new algorithms appear that position themselves as an "improvement of all previous ones", a not bad idea is to test their practical applicability in terms of the time unit "AU" (as demonstrated in Section 6). And in case that 1AU appears to be insufficient to find a solution by the new method, this clearly indicates that one should not rush into programming such an algorithm.

The answer to the second question is ambiguous and requires a detailed analysis of each algorithm under consideration. It is only clear that the mere existence of "excellent theoretical performance guarantees" for an algorithm does not guarantee equally excellent practical behavior of this algorithm in solving real-size problems (as shown by the above analysis of the algorithms from (Shmoys et al., 1994) and (Jansen et al., 2003)). It is obvious that all three algorithms were not originally intended for solving practical instances of the Job Shop problem. They have purely **theoretical significance** and contribute to the **theoretical analysis of the parametric complexity of an approximate solution** of this problem.

Thus, our algorithm remains today **the only workable algorithm** for an approximate solution to the Job Shop problem with **theoretically justified a priori bounds** of their quality, enabling one for **practically significant instances** of this problem to find solutions with acceptable accuracy and in a realistic time.

As for the approach we have developed for an approximate solution of the Job Shop problem (first proposed in (Sevastyanov, 1984, 1986) and improved in this paper), its potential, as we see it, is far from being exhausted and lies in two possible directions. First, we are not sure that this paper presents the most perfect method for reducing the original problem to a compact vector summation problem. **Reducing the degree of dependence of the solution error on parameters** m and μ is, in our opinion, a relevant and quite achievable goal. Second, the unimprovability of the (presented here) accuracy bound of the vector summation algorithm **had not been proved** either, which gives hope for a possibility of its further improvement.

References

[1] Babuskin, A.I. Basta, A.L., and Belov, I.S. *The scheduling a three-machine job shop problem*, Avtomat. i Telemeh., No. 7 (1976), 154–158 (in Russian).

[2] Babushkin, A.I., Bashta, A.A., and Belov, I.S. *Scheduling for problems of counterroutes*, Cybern. Syst. Anal. **13** (1977), 611–617; doi: 10.1007/BF01069562

[3] Babuskin, A.I., Basta, A.L., Belov, I.S., and Dushin, B.I. *Minimizing of the production line cycle*, Avtomat. i Telemeh., No. 6 (1975), 161–167 (in Russian).

[4] Belov, I.S. and Stolin, Ja.N. *An Algorithm for the Single-Route Scheduling Problem*, in: Mathematical Economics and Functional Analysis, "Nauka", Moscou (1974), 248–257 (in Russian).

[5] Chen, B., Potts, C.N., and Woeginger, G.J. *A review of machine scheduling: complexity, algorithms and approximability*, in: Handbook of Combinatorial Optimization, Boston: Kluwer Acad. Publ., **3** (1998), 21–169; doi: 10.1007/978-1-4613-0303-9_25

[6] Dushin, B.I. *Algorithm for the p-route Johnson problem*, Kibernetika, 2 (1989), 119–122 (in Russian).

[7] Garey, M.R., Johnson, D.S., and Sethi, R. *The Complexity of Flowshop and Jobshop Scheduling*, Math. Oper. Res. **1** (1976), 117–129; doi: 10.1287/moor.1.2.117

[8] Klaus Jansen, Roberto Solis-Oba, and Maxim Sviridenko (2003) Makespan Minimization in Job Shops: A Linear Time Approximation Scheme, SIAM J. Discret. Math., 16(2), 288–300. doi: 10.1137/S0895480199363908

[9] A. Kononov, S. Sevastianov, and I. Tchernykh, When difference in machine loads leads to efficient scheduling in open shops, Annals of Operations Research, 92 (1999), 211-239. DOI:10.1023/A:1018986731638

[10] Lenstra, J.K., Rinnooy Kan, A.H.G., and Brucker, P. *Complexity of machine scheduling problems*, Annals of Oper. Res. **1** (1977), 343–362; doi: 10.1016/S0167-5060(08)70743-X

[11] Neumytov, Ju.D., and Sevastyanov, S.V. *Approximation algorithm with a tight accuracy bound for the three-machine counter-routes problem*, Upravlyaemye Sistemy **31** (1993), 53–65 (in Russian).

[12] Sevastyanov, S.V. *Asymptotical Approach to Some Scheduling Problems*, in: Third All-Union Conf. Problems of Theoretical Cybernetics. Thes. Dokl., Novosibirsk (June, 1974), 67–69 (in Russian).

[13] S.V. Sevastyanov, *Asymptotical Approach to Some Scheduling Problems*, Upravlyaemye Sistemy **14** (1975), 40–51 (in Russian).

[14] S.V. Sevastyanov (1980) Approximate Solution to a Calendar-Planning Problem, *Upravlyaemye Sistemy* **20**, 49—63 (in Russian)

[15] Sevastyanov, S.V. *Some generalizations of the Johnson problem*, Upravlyaemye Sistemy **21** (1981), 45–61 (in Russian).

[16] Sevastyanov, S.V. *Algorithms with performance guarantees for the Johnson and the Akers-Friedman problems in the case of three machines*, Upravlyaemye Sistemy **22** (1982), 51–57 (in Russian).

[17] S.V. Sevast'yanov, Efficient construction of schedules close to optimal for the cases of arbitrary and alternative routes of parts, Soviet Math. Dokl., **29** (1984), 447–450; Zbl 0586.90051

[18] Sevast'yanov, S.V. *Bounding algorithm for the routing problem with arbitrary paths and alternative servers*, Cybern. Syst. Anal. **22** (1986), 773–781; doi: 10.1007/BF01068694
(Translated from "Kibernetika", November-December, No.6, (1986), 74–79.)

[19] Sevastyanov, S.V., *On the Compact Vector Summation*, Diskretnaya Matematika, **3**(3) (1991), 66–72. (in Russian)

[20] Sevastyanov, S.V. *Construction of an approximate schedule for a flow-type system*, Upravlyaemye Sistemy, **31** (1993), 66–71 (in Russian).

[21] Sevast'janov, S.V. *On Some Geometric Methods in Scheduling Theory: a survey*, Discrete Appl. Math. **55**(1) (1994), 59–82;
doi: 10.1016/0166-218X(94)90036-1

[22] Sevast'janov, S.V. *Vector summation in Banach space and polynomial algorithms for flow shops and open shops*, Mathematics of Operations Research, **20**(1) (1995), 90–103; doi: 10.1287/moor.20.1.90

[23] Sevast'yanov, S.V. *Efficient scheduling in open shops*, in: A.D. Korshunov (ed.), Discrete Analysis and Oper. Res., Kluwer, Dordrecht, 1996, 235–255;
doi: 10.1007/978-94-009-1606-7_17

[24] Sevast'yanov, S.V. *Nonstrict vector summation in the plane and its application to scheduling problems*, in: A.D. Korshunov (ed.), Operations Research and Discrete Analysis, Kluwer, Dordrecht, 1997, 241–272;
doi: 10.1007/978-94-011-5678-3_19

[25] Sevastianov, S., *Nonstrict vector summation in multi-operation scheduling*, Annals of Operations Research, **83** (1998), 179–211;
doi: 10.1023/A:1018908013582

[26] D.B. Shmoys, C. Stein and J. Wein, *Improved Approximation Algorithms for Shop Scheduling Problems*, SIAM Journal on Computing **23** (1994), 617–632;
doi: 10.1137/S009753979222676X

[27] Sotskov, Y.N., and Shakhlevich, N.V. *NP-hardness of shop-scheduling problems with three jobs*, Discrete Applied Math. 59 (1995), 237–266;
doi: 10.1016/0166-218X(95)80004-N

Appendix A. Searching for a base of a given family of vectors

Next, $\mathrm{lin}(Y)$ will denote the linear hull of a set $Y \subseteq \mathbb{R}^d$.

Definition 4. A set of vectors $X_B \subseteq \mathbb{R}^d$ is called a *base of a family of vectors* $X \subset \mathbb{R}^d$, if X_B is linearly independent and $X_B \subseteq X \subseteq \mathrm{lin}(X_B)$.

The following lemma was proved in [14].

Lemma 6 ([14])**.** *Let a family of vectors $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$ and a family of numbers $\Lambda = \{\lambda_j \in [0,1] \mid j \in [n]\}$ be given. Then there exists an algorithm that in $O(nd^2)$ time finds a base X_B of family X and a family of numbers $\Lambda' = \{\lambda'_j \in [0,1] \mid j \in [n]\}$ such that*

$$\left(\sum_{j\in[n]} \lambda'_j x_j = \sum_{j\in[n]} \lambda_j x_j \doteq x \right) \ \& \ \left(\{x_j \mid \lambda'_j \in (0,1)\} \subseteq X_B \right).$$

It is easy to verify the existence of the desired base of the family of vectors X and of the family of numbers $\{\lambda'_j\}$. Indeed, if the family of vectors $\{x_j \mid \lambda_j \in (0,1)\}$ is linearly dependent, then we can find this dependence and, using it, change the coefficients $\{\lambda_j \in (0,1)\}$ while preserving the sum $\sum \lambda_j x_j$ so that at least one of the coefficients turns to 0 or 1. Repeating this procedure no more than n times, we arrive at a linearly independent family of vectors $\{x_j \mid \lambda_j \in (0,1)\}$, which is obviously contained in some base of family X. If (in the case of $|X| > d$) we first consider subfamilies of vectors $X' \subseteq X$ of cardinality $|X'| = d+1$, and use the usual Gauss method to find (in $O(d^3)$ time) the coefficients of the linear dependence of vectors in each such subfamily, then we can obtain the desired base and the family of numbers $\{\lambda'_j\}$ in time $O(nd^3)$. The procedure below yields the desired result in $O(nd^2)$ time.

The procedure executes a loop on $t = 1, \dots, n$. At the end of each iteration t of this loop, we know coefficients $\{\lambda'_j \in [0,1] \mid j \in [n]\}$ and a partition of the index set $[n]$ into three subsets: $[n] = J_Z \cup J_W \cup J_R$ such that:

$$\sum_{j\in[n]} \lambda'_j x_j = x; \ \lambda'_j = \lambda_j, \ \forall\, j \in J_R = \{t+1, \dots, n\}; \ \lambda'_j \in \{0,1\}, \ \forall\, j \in J_Z;$$

$$X_B = \{x_j \mid j \in J_W\} \text{ is a base of the subfamily } \{x_j \mid j \in [t] = J_W \cup J_Z\}.$$

Thus, initially all coefficients λ'_j coincide with λ_j. At subsequent iterations t, the set of indices $\{j \in J_R\}$ corresponds to those coefficients λ'_j that *have not yet been transformed*, and the set J_Z corresponds to those ones that *will not be further transformed* (they are integers). The coefficients $\{\lambda_j \mid j \in J_W\}$ are "in progress".

The base X_B changes dynamically during the iterations of the loop over t: new vectors are added to it, while some vectors (that were previously included) leave it. (In the latter case, their indices move from set J_W to set J_Z.) The current list of the base is determined by the set $J_W = \{j'(1), j'(2), \dots, j'(s)\}$ of indices of the vectors included in it. It should be noted that those indices of the base vectors will not necessarily be listed in ascending order, since the *place of each new vector in the base* is dynamically determined during the algorithm. In this case, the place in the base that a vector receives when it enters the base is **permanent** until the moment of its leaving the base or until the end of the algorithm. This place corresponds to the column number of matrix C of the transition from base $X_B = \{x_{j'(1)}, \dots, x_{j'(s)}\}$ to basis B (to be defined below).

Thus, at each iteration t, beyond the sets J_Z, J_W, J_R, we adjust the following:
— basis $B = \{b_1, \dots, b_s\}$ of space $\mathrm{lin}\{x_1, \dots, x_t\}$ and the first $s = |J_W|$ components of the formed permutation $\pi = (\pi_1, \dots, \pi_d)$ of coordinates of space $\mathbb{R}^d$, so that $b_k(\pi_i) = 0, \ \forall\, i \in [k-1]; \ b_k(\pi_k) \neq 0, \ \forall\, k \in [s]$. Namely, both basis B and permutation π may be incremented by a new, $(s+1)$-th element, while s previously

defined elements remain unchanged;
— matrix $C = (c_{k,i})_{k,i\in[s]}$ of the transition from base $X_B = \{x_{j'(1)}, \dots, x_{j'(s)}\}$ to basis B:

$$b_k = \sum_{i\in[s]} c_{k,i}\, x_{j'(i)},\ k \in [s]; \tag{30}$$

— function $j'(i)$, specifying the position of each vector $x_j \in X_B$ in the base and in matrix $(c_{k,i})$ (the i-th column of this matrix corresponds to vector $x_{j'(i)}$).

When adding a new vector x_t to consideration at iteration t, the following three cases are possible.

(1) Vectors $X_B \cup \{x_t\}$ are linearly independent. In this case, vector x_t is added to basis X_B and placed in it at the $(s+1)$-th position, where $s = |X_B|$. In addition, basis B and permutation π are supplemented with $(s+1)$-th elements; matrix C is incremented by one more column and one row.

(2) Vectors $X_B \cup \{x_t\}$ are linearly dependent; vector x_t does not get into the base. After recalculating the coefficients $\{\lambda'_j\}$, index t of vector x_t is transferred from J_W to J_Z. The base, the basis, matrix C, and permutation π remain unchanged.

(3) Vectors $X_B \cup \{x_t\}$ are linearly dependent; one of vectors of the base $(x_{j'(i^*)})$ is removed from it, and vector x_t is inserted into the base **at the vacated position** i^*. (In matrix C it will correspond to the recalculated i^*-th column.) Matrix C and function $j'(i)$ are recalculated; basis B and permutation π remain unchanged.

```
                    **Procedure** *Search_for_a_Base*(**integer**: $n,s$;
**family of integers:** $\{j_B[i] \mid i \in [d]\}$; **family of vectors in** $\mathbb{R}^d$: $X=\{x_t \mid t \in [n]\}$;
    **family of reals:** $\Lambda = \{\lambda_j \in [0,1] \mid j \in [n]\}, \Lambda' = \{\lambda'_j \in [0,1] \mid j \in [n]\}$);
% The OUTPUT parameter $\{j_B[i] \mid i \in [s]\}$ is the set of indices of the vectors
% $x_t \in X$ forming the base $X_B$ of $X$ that we are looking for
% **Local parameters:**
  **integer:** $t, k, j, i, i^*$; **real:** $\varepsilon, \varepsilon^*$; **vector in** $\mathbb{R}^d : b$;
  **family of integers:** $J_R, J_Z, J_W, J'_W, \{\pi_k \mid k \in [d]\}$;
  **family of reals:** $\{\mu_k \mid k \in [n]\}$; $\{\mu'_k \mid k \in [n]\}$; $\{c_{k,i} \mid k, i \in [d]\}$;
  **family of vectors in** $\mathbb{R}^d$: $B=\{b_k \mid k \in [d]\}$;
**BEGIN**
  $J_R \leftarrow [n];\ J_Z \leftarrow \varnothing;\ J_W \leftarrow \varnothing;\ \lambda'_j \leftarrow \lambda_j,\ \forall\, j \in [n];\ s \leftarrow 0;$
  **for** $t \leftarrow 1, \dots, n$ **do begin**
    $J_R \leftarrow J_R \setminus \{t\}$; **if** $x_t = 0$ **then** $\{\lambda'_t \leftarrow 0;\ J_Z := J_Z \cup \{t\}\}$
    **else begin**
      $J_W \leftarrow J_W \cup \{t\};\ j_B[s+1] \leftarrow t;\ b \leftarrow x_t;$
      **for** $k \leftarrow 1, \dots, s$ **do** $\{\mu_k \leftarrow b(\pi_k)/b_k(\pi_k);\ b \leftarrow b - \mu_k b_k\}$;
    % We have obtained vector $b = x_t - \sum_{k\in[s]} \mu_k b_k$ such that $b(\pi_i) = 0$ $(i \in [s])$.
    % Substituting the values of $\{b_k\}$ from (30) to this expression, we obtain
    % $b = -\sum_{i\in[s+1]} \mu'_{j_B[i]} x_{j_B[i]}$, where the coefficients $\{\mu'_{j_B[i]}\}$ are defined as:
      $\mu'_{j_B[i]} \leftarrow \sum_{k\in[s]} \mu_k \cdot c_{k,i},\ i \in [s];\ \mu'_{j_B[s+1]} \leftarrow -1;$
% As proved in [14], $x_{j_B[s+1]}$ linearly depends on vectors $\{x_{j_B[1]}, \dots, x_{j_B[s]}\}$, if and
% only if $b = 0$. We next check, if $b(\pi_i) = 0$ holds for all $i \in [d] \setminus \{\pi_i \mid i \in [s]\}$.
      **if** $b \neq 0$ **then begin**
        find a coordinate $i^* \in [d] \setminus \{\pi_i \mid i \in [s]\}$ such that $b(i^*) \neq 0$;
        $\pi_{s+1} \leftarrow i^*;\ b_{s+1} \leftarrow b;$
      % We next complement matrix $(c_{i,j})$ by defining the coefficients from the
      % $(s+1)$-th row and the $(s+1)$-th column as:
        $c_{s+1,i} \leftarrow -\mu'_{j_B[i]},\ i \in [s+1]$;
        $c_{k,s+1} \leftarrow 0,\ k \in [s];\ s \leftarrow s+1$;
      end % “if $b \neq 0$”
      else begin % the case when vectors $\{x_{j_B[1]}, \dots, x_{j_B[s+1]}\}$ are linearly dependent,
                  % since $\sum_{i\in[s+1]} \mu'_{j_B[i]} x_{j_B[i]} = 0$, while $\mu'_{j_B[s+1]} = -1$.
        Define the set $J'_W \doteq \{j \in J_W \mid \mu'_{j_B[i]} \neq 0\}$;
        find $\varepsilon^* = \max\{\varepsilon \mid \lambda'_j + \varepsilon\mu'_j \in [0,1],\ \forall\, j \in J'_W\}$;
        redefine the coefficients $\{\lambda'_j \mid j \in J'_W\}$ as: $\lambda'_j \leftarrow \lambda'_j + \varepsilon^*\mu'_j,\ j \in J'_W$;
      % At that, the sum $\sum_{j\in[n]} \lambda'_j x_j$ does not change and at least one of the
      % coefficients $\{\lambda'_j \mid j \in J'_W\}$ takes a value from $\{0,1\}$.
        Find $i^* \in [s+1]$ such that $(\lambda'_{j_B[i^*]} \in \{0,1\})\ \&(j_B[i^*] \in J'_W)$;
        $J_W \leftarrow J_W \setminus \{j_B[i^*]\};\ J_Z \leftarrow J_Z \cup \{j_B[i^*]\}$;
      % In case that $j_B[i^*] = j_B[s+1] = t$, there is nothing to do next, because
      % both the base and the basis **remain unchanged**. Otherwise, we replace
      % vector $x_{j_B[i^*]}$ of base $X_B$ by vector $x_t$ and recalculate the coefficients $(c_{k,i})$
      % of the transition from the base to basis $B$ (which remains unchanged)
      % according to formula (30). (We note here that $J''_W \doteq J'_W \setminus \{t\} \neq \varnothing$, since
      % $x_t = \sum_{j\in J''_W} \mu'_j x_j \neq 0$.) So,
        if $i^* \leq s$ then begin
          $c_{k,i^*} \leftarrow c_{k,i^*}/\mu'_{j_B[i^*]}$; $c_{k,i} \leftarrow c_{k,i} - c_{k,i^*}\mu'_{j_B[i]},\ i \in [s] \setminus \{i^*\},\ k \in [s]$;
          $j_B[i^*] \leftarrow t$;
        end; % “if”
      end; % “else”
  end; % “for $t$”
END; % of Procedure Search_for_a_Base
```

Appendix B. Vector summation algorithm $\mathcal{A}_1$ from Theorem 1

Definition 5. Let B be a convex set in $\mathbb{R}^d$, I be a finite set such that $k \doteq |I| \geq d$, $a \in \mathbb{R}^d$, $B_a \doteq \operatorname{conv}\{0, a - B/d\}$. A family of vectors $\{x_i \mid i \in I\} \subset B$ is called a *B_a-balanced system of vectors* (*B_a-BSV*, for short), if there is a family of numbers $\{\lambda_i \in \mathbb{R} \mid i \in I\}$ satisfying the conditions:

$$\lambda_i \in [0,1],\ i \in I, \tag{31}$$

$$\sum_{i\in I} \lambda_i = k - (d-1), \tag{32}$$

$$\sum_{i\in I} \lambda_i x_i \in B_a. \tag{33}$$

The construction of the desired sequence π^* of summing the vectors of family $X = \{x_i \mid i \in [N]\} \subset \mathbb{R}^d$ by algorithm $\mathcal{A}_1$ is based on the cyclic usage of the procedure *Next_BSV* (described below), which, heaving at its input a B_a-balanced system of vectors $U = \{x_i \mid i \in I\} \subset \mathbb{R}^d$ (defined for a subset $I \subseteq [N]$, s.t. $|I| \geq d+1$), finds an index $i^* \in I$ such that the family $U \setminus \{x_{i^*}\}$ is a B_a-BSV again.

The correctness of this procedure was justified in

Lemma 7 ([19]). *Let B be a convex set in $\mathbb{R}^d$, $a \in \mathbb{R}^d$, $\{x_i \in B \mid i \in I\}$ be a B_a-BSV with $|I| \doteq k \geq d+1$, for which the corresponding family of numbers $\{\lambda_i \mid i \in I\}$ satisfying the conditions (31)–(33) is known. Then there exists an index $i^* \in I$ such that the system of vectors $U' = \{x_i \mid i \in I \setminus \{i^*\}\}$ is B_a-BSV again; index i^* and the numbers $\{\lambda'_i\}$ corresponding to vectors from U' can be found in $O(kd^2)$ time.*

```
  Procedure Next_BSV(integer: $i^*, k$; finite set of integers: $I$;  % $|I| = k$
 family of vectors in $\mathbb{R}^d$: $U = \{x_i \mid i \in I\}$; family of reals: $\Lambda = \{\lambda_i \mid i \in I\}$);
% The input family of vectors $U$ is a $B_a$-BSV with given coefficients $\{\lambda_i \mid i \in I\}$.
% The set $I$ (with the output value $I \leftarrow I \setminus \{i^*\}$), numbers $i^*$ and $k$, and the
% transformed families $U$ and $\Lambda$ are also output parameters of the procedure.
% Local parameters:
    integer: $i, i_0$;
    real: $\varepsilon, r$;
    a finite set of integers: $I_1, I_W, I_W^-, I_W''$;
    a family of reals with indices $\{i \in I\}$: $\{\lambda'_i\}, \{\lambda''_i\}, \{\mu_i\}, \{\mu'_i\}, \{\mu''_i\}, \{\eta_i\}$;
    family of vectors in $\mathbb{R}^{d+1}$: $\bar{U} = \{\bar{x}_i \mid i \in I\}$; $\bar{U}' = \{\bar{x}_i \mid i \in I_W\}$;
    vector in $\mathbb{R}^d$: $c, b$;
    vector in $\mathbb{R}^{d+1}$: $\bar{c}, \bar{a}$;
BEGIN
```

Put $\lambda'_i \leftarrow (k-d)\lambda_i/(k-d+1)$, $\forall\, i \in I$. Then we have:

$$\lambda'_i \in [0,1],\ i \in I, \tag{34}$$

$$\sum_{i \in I} \lambda'_i = k - d, \tag{35}$$

$$\sum_{i \in I} \lambda'_i x_i \in B_a. \tag{36}$$

To obtain the desired result (a B_a-BSV with the number of vectors less by one), it remains to transform the coefficients $\{\lambda'_i\}$ while preserving the properties (34)–(36) so that one of the coefficients (λ'_{i^*}) becomes zero (which would mean that the family of vectors $U' = \{x_i \mid i \in I \setminus \{i^*\}\}$ becomes a B_a-BSV). The desired transformation will require at most four “Steps” described below.

Step 1. Let us define a family of vectors $\bar{U} = \{\bar{x}_i \doteq (x_i, 1) \mid i \in I\} \subset \mathbb{R}^{d+1}$, and then transform the coefficients $\{\lambda'_i\}$ while preserving the condition (34) and the sum $\sum_{i \in I} \lambda'_i \bar{x}_i$ so that the subfamily of vectors $\bar{U}' \doteq \{\bar{x}_i \in \bar{U} \mid \lambda'_i \in (0,1)\}$ is contained in some base $\bar{U}^*$ of family $\bar{U}$. By Lemma 6, the desired coefficients $\{\lambda'_i\}$ and the base $\bar{U}^* \subseteq \bar{U}$ can be found (by means of the procedure *Search_for_a_Base* described in Application A) in $O(|I|d^2)$ time. If among the coefficients $\{\lambda'_i\}$ there is a coefficient $\lambda'_{i^*} = 0$, then the family of vectors $\{x_i\}_{i \neq i^*}$ is the desired B_a-BSV with coefficients $\{\lambda_i \leftarrow \lambda'_i\}_{i \neq i^*}$. Next we assume that

$$\lambda'_i > 0,\ \forall\, i \in I. \tag{37}$$

Let $I_W \doteq \{i \in I \mid \lambda'_i \in (0,1)\}$, $I_1 \doteq \{i \in I \mid \lambda'_i = 1\}$. Then using (35) and (37), we obtain

$$k - d = \sum_{i \in I} \lambda'_i = \sum_{i \in I_1} \lambda'_i + \sum_{i \in I_W} \lambda'_i = |I_1| + \sum_{i \in I_W} \lambda'_i = k - |I_W| + \sum_{i \in I_W} \lambda'_i,$$

implying $d+\sum_{i\in I_W}\lambda'_i=|I_W|\le|\bar U^*|\le d+1$. Thus, $\sum_{i\in I_W}\lambda'_i$ must be an integer not greater than 1. It cannot be 0 (since $|I_W|\ge d$), so, we obtain:

$$\sum_{i\in I_W}\lambda'_i=1,\ |I_W|=d+1,\ \bar U' \text{ is a basis of } \mathbb{R}^{d+1}.$$

Let us transform the coefficients $\{\lambda'_i\,|\,i\in I_W\}$ while preserving the properties (34)–(36) so that one of the coefficients becomes zero. To that end, we denote:

$$c\doteq-\sum_{i\in I_1}x_i,\ \bar c\doteq(c,1)\in\mathbb{R}^{d+1},\ b\doteq\sum_{i\in I}\lambda'_ix_i\in B_a. \tag{38}$$

Step 2. Let us find the unique representation $\bar c=\sum_{i\in I_W}\mu_i\bar x_i$. The coefficients $\{\mu_i\}$ can be found, for example, by the Gauss method in $O(d^3)\le O(kd^2)$ time. If

$$I_W^-\doteq\{i\in I_W\,|\,\mu_i\le0\}\ne\varnothing, \tag{39}$$

we define $\varepsilon\doteq\max_{i\in I_W^-}\{-\mu_i/(\lambda'_i-\mu_i)\}$. Clearly, $\varepsilon\in[0,1)$,

$$\lambda''_i\doteq(1-\varepsilon)\mu_i+\varepsilon\lambda'_i=\mu_i+\varepsilon(\lambda'_i-\mu_i)\ge0,\ \forall\ i\in I_W,$$

and there is an $i^*\in I_W^-$ such that $\lambda''_{i^*}=0$. It follows from $\sum_{i\in I_W}\mu_i=1,\sum_{i\in I_W}\lambda'_i=1$ that $\sum_{i\in I_W}\lambda''_i=1$. Put $\lambda''_i\leftarrow1,\ \forall\ i\in I_1$. Then $\sum_{i\in I}\lambda''_i=|I_1|+\sum_{i\in I_W}\lambda''_i=k-|I_W|+1=k-(d+1)+1=k-d$. Furthermore, due to (38) and $\varepsilon\in[0,1)$,

$$\begin{aligned}\sum_{i\in I}\lambda''_ix_i &= \sum_{i\in I_1}x_i+\sum_{i\in I_W}\lambda''_ix_i=-c+\sum_{i\in I_W}[(1-\varepsilon)\mu_i+\varepsilon\lambda'_i]x_i=\\ &= -c+(1-\varepsilon)c+\varepsilon(c+b)=\varepsilon b\in B_a.\end{aligned}$$

Thus, the family of vectors $U=\{x_i\,|\,i\in I'\}$ (specified for the set of indices $I'\doteq I\setminus\{i^*\}$) is a B_a-BSV with coefficients $\{\lambda_i\leftarrow\lambda''_i\,|\,i\in I'\}$.

Let (39) not hold, which means, $\mu_i\in(0,1),\ \forall\ i\in I_W$.

Step 3. Find the unique representation of vector $\bar a=(da,1)\in\mathbb{R}^{d+1}$ in the form: $\bar a=\sum_{i\in I_W}\eta_i\bar x_i$. Since $\sum_{i\in I_W}(d\mu_i+\eta_i)=d+1$, there exists $i_0\in I_W$ such that $d\mu_{i_0}+\eta_{i_0}\le1$. Since $\mu_{i_0}>0$, the inequality $\eta_{i_0}<1$ holds, and we can define

$$r\doteq\frac{\mu_{i_0}}{1-\eta_{i_0}}\le\frac{1}{d}. \tag{40}$$

As shown in [19], vector $c-rx_{i_0}$ can be presented as

$$c-rx_{i_0}=\sum_{i\in I_W}\mu'_ix_i-dar \tag{41}$$

with coefficients $\mu'_i\doteq\mu_i+\mu_{i_0}\eta_i/(1-\eta_{i_0}),\ i\in I'_W\doteq I_W\setminus\{i_0\};\ \mu'_{i_0}\doteq0$. Thus,

$$\sum_{i\in I_W}\mu'_i=\sum_{i\in I'_W}\mu'_i=\sum_{i\in I'_W}\mu_i+\frac{\mu_{i_0}}{(1-\eta_{i_0})}\sum_{i\in I'_W}\eta_i=\sum_{i\in I'_W}\mu_i+\mu_{i_0}=\sum_{i\in I_W}\mu_i=1,$$

so, putting $\mu'_i=1$ $(i\in I_1)$, we obtain $\sum_{i\in I}\mu'_i=k-(d+1)+1=k-d$. Using (41) and (40), we also get

$$\sum_{i\in I}\mu'_ix_i=\sum_{i\in I_1}\mu'_ix_i+\sum_{i\in I_W}\mu'_ix_i=-c+c-rx_{i_0}+dar=dr\left(a-\frac{1}{d}x_{i_0}\right)\in B_a$$

(since $a - \frac{1}{d}x_{i_0} \in a - \frac{1}{d}B \subseteq B_a$ and $dr \le 1$). Thus, if $\mu'_i \ge 0,\ \forall\ i \in I_W$, we put $I' \doteq I \setminus \{i_0\},\ \lambda_i \leftarrow \mu'_i\ (i \in I'),\ i^* \leftarrow i_0$, and the family of vectors $U = \{x_i \,|\, i \in I'\}$ becomes a B_a-BSV with coefficients $\{\lambda_i \,|\, i \in I'\}$.

We next suppose that $I''_W \doteq \{i \in I_W \,|\, \mu'_i < 0\} \neq \varnothing$.

Step 4. Define $\varepsilon \doteq \max_{i \in I''_W}\{-\mu'_i/(\mu_i - \mu'_i)\}$. Then $\varepsilon \in (0,1)$ (because all $\mu_i > 0$). So, we get $\mu''_i \doteq (1-\varepsilon)\mu'_i + \varepsilon\mu_i = \mu'_i + \varepsilon(\mu_i - \mu'_i) \ge 0,\ \forall\ i \in I_W$,

$$\exists\ i^* \in I''_W :\ \mu''_{i^*} = 0,$$
$$\textstyle\sum_{i \in I_W} \mu''_i = 1.$$

Putting $\mu''_i = 1\ (i \in I_1)$, we have $\sum_{i \in I} \mu''_i = k - d$. Then, using (41), the definition of $\{\mu_i\}$ (see Step 2), and (38), we obtain:

$$\begin{aligned}\textstyle\sum_{i \in I} \mu''_i x_i = \sum_{i \in I_1} x_i + \sum_{i \in I_W} \mu''_i x_i = -c + \sum_{i \in I_W}[(1-\varepsilon)\mu'_i + \varepsilon\mu_i]x_i = \\ = -c + (1-\varepsilon)(c - r x_{i_0} + dar) + \varepsilon c = (1-\varepsilon)dr\left(a - \tfrac{1}{d}x_{i_0}\right) \in B_a.\end{aligned}$$

Thus, the family of vectors $U \doteq \{x_i \,|\, i \in I'\}$ (for $I' \doteq I \setminus \{i^*\}$) is a B_a-BSV with coefficients $\{\lambda_i \leftarrow \mu''_i \,|\, i \in I'\}$.

Put $I \leftarrow I'$.
END; % of Procedure *Next_BSV*

Algorithm $\mathcal{A}_1(d, a, N, X; \pi)$
(of finding the order π of a "compact summation" of the family of vectors $X = \{x_i \,|\, i \in [N]\} \subset \mathbb{R}^d,\ \sum_{x_i \in X} x_i = 0$)

% **Notation** (see Theorem 1, page 9, and Definition 5, page 37):
% $B \doteq \mathrm{conv}\{X\};\ a \in \mathbb{R}^d;\ B_a \doteq \mathrm{conv}\{0, a - B/d\}$
% To obtain the desired permutation π of indices $k \in [N]$, we find a sequence of
% sets $I_k \subseteq [N]$ $(k = 1, \dots, N)$ such that

$$I_1 \subset I_2 \subset \cdots \subset I_N = [N];\quad |I_k| = k,\ \forall\ k \in [N]; \tag{42}$$

$$\sum_{i \in I_k} x_i \in (d-1)B + B_a,\ \forall\ k \in [N]. \tag{43}$$

% To that end, we first observe that **if** a family of vectors $X(I) \doteq \{x_i \,|\, i \in I\} \subset \mathbb{R}^d$
% (for $|I| \ge d$) is a B_a-BSV with coefficients $\{\lambda_i \,|\, i \in I\}$, **then** we have:

$$\sum_{i \in I} x_i = \sum_{i \in I}(1-\lambda_i)x_i + \sum_{i \in I}\lambda_i x_i \in \sum_{i \in I}(1-\lambda_i)B + B_a = (d-1)B + B_a,$$

% which means that $X(I)$ meets (43). Also observe that $X(I_N)$ is a B_a-BSV with
% coefficients $\{\lambda_i = (N-(d-1))/N \,|\, i \in I_N\}$, since $\lambda_i \in [0,1],\ \sum \lambda_i = N - (d-1)$,
% $\sum \lambda_i x_i = 0 \in B_a$. Applying these two facts and Procedure *Next_BSV*, we can
% find subsets $I_d, \dots, I_{N-1}$ which are B_a-BSV (and thus, meet (43)) and can define
% the items $\pi_k \in I_k \setminus I_{k-1}$ $(k = d+1, \dots, N)$ of permutation π, respectively. Finally,
% defining the subsets $\{I_k \,|\, k < d\}$ satisfying (42) arbitrarily, we obtain for them:

$$\frac{1}{d-1}\sum_{i \in I_k} x_i = \frac{k}{d-1}\sum_{i \in I_k}\frac{1}{k}x_i \doteq \frac{k}{d-1}x' \in B,$$

% since $x' \doteq \sum_{i \in I_k}\frac{1}{k}x_i \in B,\ \frac{k}{d-1} \le 1$, and $\mathbf{0} \in B$. Thus,

$$\sum_{i \in I_k} x_i \in (d-1)B \subseteq (d-1)B + B_a$$

```
% (since $\mathbf{0} \in B_a$), providing (43) again.
% This yields the following simple scheme for Algorithm $\mathcal{A}_1$.

% **Local parameters:**
   **integer:** $k, i^*$;

**BEGIN**
   $I \leftarrow [N];\ \lambda_i \leftarrow (N-(d-1))/N,\ \forall\, i \in I$;
   **for** $k \leftarrow N$ **step** $-1$ **down to** $d+1$ **do**
      $\{Next\_BSV(i^*;\ I;\ U = \{x_i \in X \mid i \in I\};\ \Lambda = \{\lambda_i \mid i \in I\});\ \pi_k \leftarrow i^*\}$;
   **for** $k \leftarrow 1, \dots, d$ **do** $\{$Find $i^* \in I;\ \pi_k \leftarrow i^*;\ I \leftarrow I \setminus \{i^*\}\}$
**END.**
```

Sergey Vasilyevich Sevastyanov
Sobolev Institute of Mathematics,
pr. Koptyuga, 4,
630090, Novosibirsk, Russia
Email address: seva@math.nsc.ru