S©MR ISSN 1813-3304

СИБИРСКИЕ ЭЛЕКТРОННЫЕ
МАТЕМАТИЧЕСКИЕ ИЗВЕСТИЯ

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

Том 19, №2. стр. 601–612 (2022) УДК 519.854.2
DOI 10.33048/semi.2022.19.050 MSC 90B35

Special issue: International conference «Mathematical Optimization Theory and Operations Research» (MOTOR 2022), July 2–6, 2022, Petrozavodsk, Republic of Karelia, Russia

MINIMIZING TOTAL COMPLETION TIME ON PARALLEL MACHINES WITH UNIT LENGTH JOBS THAT NEED ONE ADDITIONAL RESOURCE

V.A.STRUSEVICH

Abstract. The paper considers a scheduling problem on parallel identical machines to minimize the sum of the completion times. All jobs are of unit length and require one unit of one of the additional resources. We offer an algorithm that is much faster and simpler than previously available.

Keywords: resource constrained scheduling, unit length jobs, total completion time, parallel identical machines.

1. Introduction

Within Machine Scheduling there is an established research area known as *Scheduling under Resource Constraints*. There are multiple publications in this area which have been reviewed in a number of surveys, from the earliest [2] to the most recent [3]. The models with resource constraints extend the classical Machine Scheduling models, traditionally formulated in terms of jobs to be processed on machines, by allowing additional resources to be present. These resources are different from the processing machines, but their use is compulsory for processing.

In this paper, we consider scheduling problems on parallel machines under specific resource constraints. The jobs of set $N = \{J_1, J_2, \dots, J_n\}$ have to be

Strusevich, V.A., Minimizing total completion time on parallel machines with unit length jobs that need one additional resource.

Received April, 27, 2022, published August, 30, 2022.

processed on m identical parallel machines $M_1, \dots, M_m$. The processing time of job $J_j \in N$ is equal to p_j. There are also Q renewable resources, so that at any time moment exactly one unit of each resource is available. Some jobs, known as the resource jobs, at any time of their processing must consume one unit of exactly one of these Q resources. An instance may also contain n_0 jobs which do not require these resources; however, it is convenient to think of any such a job as a job that consumes one unit of a unique additional resource r, $Q+1 \leq r \leq Q+n_0$. Such an interpretation allows us to denote $q = Q + n_0$ as the total number of resources and to treat all jobs as the resource jobs. Therefore, set N can be seen as partitioned into q disjoint sets $N^1, \dots N^q$ such that each job of set N^r at any time of its processing requires one unit of resource r, $1 \leq r \leq q$. The resource constraints imply that no two jobs of the same set N^r can be processed simultaneously.

Resource constraints of this type have found applications and motivation in various areas. Hebrard et al. [6] motivate their study by the problem that arises in satellite data download management. Janssen et al. [7] present a motivation based on a manufacturing problem in the microelectronic industry. An example in Strusevich [12] is related to human resource management; another meaningful example is contained in this paper (see Example 1 in Section 4).

For a scheduling problem, the completion time of a job J_j in a feasible schedule S is denoted by $C_j(S)$; often, if no confusion arises, the reference to the schedule is omitted and we simply write C_j. The two most popular objective functions to be minimized are the maximum completion time or the *makespan* $C_{\max}(S) = \max\{C_j(S) | J_j \in N\}$ and the *total completion time* $F(S) = \sum_{J_j \in N} C_j(S)$.

In scheduling with resource constraints many results are obtained under the assumption that the processing times of all jobs are equal to one time unit. We follow the traditional terminology and say that these jobs are jobs of Unit Execution Time or the *UET jobs*. Among the results presented in [7] there is a polynomial-time algorithm for minimizing the total completion time $F(S)$ of the UET jobs on m parallel identical machines under the resource constraints of the described type. In this note, we develop a much faster algorithm for this problem.

The remainder of this paper is organized as follows. Section 2 introduces notation and provides a brief review of the relevant problems. Section 3 is of an auxiliary nature and discusses the problem of minimizing the total completion time on parallel machines with no resource constraints and its links to majorization of vectors. The fast algorithm for minimizing the total completion time of the UET jobs on parallel machines with resource constraints is given in Section 4. Concluding remarks and problems for further study can be found in Section 5.

2. Notation and Preliminaries

In order to denote scheduling problems in a clear and compact way, the three-field classification scheme of the form $\alpha|\beta|\gamma$ is widely accepted, where α describes a machine environment, β is responsible for presenting the processing conditions and γ is the objective function. In the case of a processing system with parallel identical machines, in the first field α we write "P"; if the number of machines is not variable (part of the input) but is fixed and equal to m, then in the first field α we write "Pm" rather than "P". In this paper, we mainly focus on the problem with the objective of the total completion time $F(S) = \sum_{J_j \in N} C_j(S)$; in this review section we also pay attention to the problems of minimizing the makespan $C_{\max}(S) = \max\{C_j(S) | J_j \in N\}$.

For the scheduling problems under the resource constraints, the notation that has become standard since its first appearance in [2] places into the middle field β a string that specifies the rules of resource usage by addressing three parameters.

Widely accepted, that notation however does not provide enough details in order to distinguish between various versions of the resource-constrained scheduling problems. Following [12], we adopt the extended scheme that adds to the middle field β a four-parameter string of the form "$res\ \rho_1\rho_2\rho_3\rho_4$", where

- ρ_1 is the number of available renewable resources;
- ρ_2 is an upper bound on the number of resources a job may need;
- ρ_3 is an upper bound on the number of units of any resource available at a time;
- ρ_4 is an upper bound on the number of units of any resource that can be consumed by a job at a time.

The value of each of these parameters is either a known constant or the symbol "$\cdot$"; in the latter case the value of the parameter is variable (part of the input). In accordance with this updated scheme, for the main resource-constrained problem considered in this paper the string "$res\ \cdot 111$" is used. This notation implies that

- $\rho_1 =$ "$\cdot$", i.e., that there several renewable resources (as above, their number is denoted by q);
- $\rho_2 = 1$, i.e., each job needs either exactly one resource at any time of its processing;
- $\rho_3 = 1$, i.e., one unit of each resource is available at a time;
- $\rho_4 = 1$, i.e., a job consumes one unit of the relevant resource at any time of its processing.

In the traditional scheme the parameter ρ_2 was missing. Therefore, in most of the previously considered models with resource constraints it was assumed that a job might need any number of resources. The four-parameter scheme is free from that drawback.

A focused review of resource-constrained scheduling on parallel machines can be found in [5]. Most of known results in the area are on the models with the unit processing times. For the problems with the UET jobs we write "$p_j = 1$" in the middle field β of the three-field notation. Some of the known results hold for a more general machine environment that identical machines, e.g, the parallel machines can be uniformly related, i.e., may have different speeds. In the latter case, in the first field α of the three-field notation scheme we write "Q" rather than "P".

Here we do not review the known results for the resource-constrained problems on parallel machines with arbitrary processing times; the relevant material is contained in the corresponding surveys [1]-[3], [5]. Neither we consider the problems for other machine environments, e.g., parallel dedicated machines [8].

The results on minimizing the makespan for the problems with the UET jobs are explained in detail in the survey [2], which also contains all relevant references that we do not quote here explicitly. Problem $P2\,|p_j = 1,\ \text{res}\ \cdot\cdot\cdot\cdot|\,C_{\max}$ with arbitrary resource constraints reduces to the matching problem and is solvable is $O\left(qn^2 + n^{5/2}\right)$ time, while each problem $P3\,|p_j = 1,\ \text{res}\ \cdot\cdot 11|\,C_{\max}$ and $Q2\,|p_j = 1,\ \text{res}\ \cdot\cdot 11|\,C_{\max}$ is NP-hard in the strong sense. Each problem

$Q2\,|p_j = 1,\ \text{res}\ 11\cdot\cdot|\,C_{\max}$ and $Q\,|p_j = 1,\ \text{res}11\cdot 1|\,C_{\max}$ is solvable in polynomial time.

In the case of minimizing the total completion time, Problem $P2\,|p_j = 1,\ \text{res}\ \cdot\cdot 11|\sum C_j$ is NP-hard in the strong sense [2]. If no more that one resource is needed by any job, the complexity status of Problem $P\,|\text{res}\ \cdot 111|\sum C_j$ with arbitrary processing times is unknown, while Problem $P\,|p_j = 1,\ \text{res}\ \cdot 111|\sum C_j$ with the UET jobs admits a polynomial-time algorithm [7]. In the remainder of this paper we present a much simpler and faster algorithm for the latter problem.

3. UET Jobs: No Resource Constraints

We start with Problem $P\,|p_j = 1|\sum C_j$ to minimize the total completion time $F(S) = \sum_{J_j\in N} C_j(S)$ of the UET jobs without resource constraints. In what follows, we assume that $n > m$, since otherwise in an optimal schedule exactly one job is processed in the interval $[0,1]$ on one of the n machines. Let $C^{[i]}(S)$ denote the latest completion time of the jobs assigned to machine M_i in a schedule S.

Recall that Problem $P\,|\,|\sum C_j$ with arbitrary processing times is solvable in $O(n\log n)$ time. The corresponding algorithm is due to Conway et al. [4] and is based on the *Shortest Processing Times (SPT)* priority rule. For completeness, we present the algorithm below.

Algorithm SPT

Step1. : Form the list of jobs by sorting them in the SPT order, i.e., in non-decreasing order of their processing times.

Step 2.: Scanning the list, assign the next job to the first available machine.

If the processing times are unit, Algorithm SPT cannot be seen as polynomial for Problem $P\,|p_j = 1|\sum C_j$, since the input of that problem is nothing else but the number of bits needed to encode the number of machines and the number of jobs. Still, it is possible to solve Problem $P\,|p_j = 1|\sum C_j$ in time that is polynomial in $O(\log n + \log m)$ by associating each job with a position (a unit time interval) on some machine. Below we reproduce the reasoning for such an algorithm, since it will be useful for deriving a fast algorithm for a more general problem with resource constraints.

For Problem $P\,|p_j = 1|\sum C_j$, define C^*, the smallest makespan, $C^* = \lceil n/m\rceil$.

Lemma 1. *In a schedule S^* that is optimal for Problem $P\,|p_j = 1|\sum C_j$ the inequality*

$$C^{[i]}(S^*) \le C^*$$

holds for each machine M_i, $1 \le i \le m$.

Proof. Suppose that in schedule S^* there is a machine M_ℓ such that $C^{[\ell]}(S^*) > C^*$. Since $\sum_{i=1}^{m} C^{[i]}(S^*) = n$, it follows from $C^* = \lceil n/m\rceil$ that there exists a machine M_k, $k \ne \ell$, such that $C^{[k]}(S^*) < C^*$. Notice that

$$C^{[\ell]}(S^*) > C^* \ge C^{[k]}(S^*) + 1.$$

Transform schedule S^* to a schedule S' by moving the last job from machine M_ℓ to become the last job on machine M_k, keeping the remaining jobs assigned as in schedule S^*. Notice that $C^{[\ell]}(S') = C^{[\ell]}(S^*) - 1$ and $C^{[k]}(S') = C^{[k]}(S^*) + 1 \leq C^*$. For schedule S' we obtain

$$F(S') = F(S^*) - C^{[\ell]}(S^*) + C^{[k]}(S^*) + 1 < F(S^*),$$

which contradicts the optimality of schedule S^*. □

Notice that in any optimal schedule S^* for at least one machine M_i the completion time $C^{[i]}(S^*)$ is equal to C^*.

Definition 1. *For Problem $P|p_j = 1|\sum C_j$, let H_n^m be the set of all vectors $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ with non-increasing positive integer components such that $x_1 = C^*$ and $\sum_{i=1}^m x_i = n$. A vector $\mathbf{x} \in H_n^m$ is called an allocation vector.*

Each allocation vector $\mathbf{x}$ defines a schedule $S(\mathbf{x})$ in which the number of jobs assigned to machine M_i is equal to x_i, $1 \leq i \leq m$. For machine M_i, its contribution to the objective function value $F(S)$ is equal to $1 + 2 + \cdots + x_i = \frac{1}{2}x_i(x_i + 1)$. Define a function $h : \mathbb{R} \to \mathbb{R}$ such that $h(u) = \frac{1}{2}u(u+1)$. It follows that

$$F(S(\mathbf{x})) = \sum_{i=1}^{m} h(x_i).$$

An allocation vector $\mathbf{y} = (y_1, y_2, \ldots, y_m)$ defines an optimal schedule $S^* = S(\mathbf{y})$ and is called an *optimal allocation vector* if the inequality

$$F(S(\mathbf{y})) \leq F(S(\mathbf{x}))$$

holds for all allocation vectors $\mathbf{x}$.

Since function $h(u)$ is convex, it is convenient to justify the choice of an optimal allocation vector by the concepts of the majorization theory [9].

Definition 2. *For two vectors $\mathbf{a} = (a_1, a_2, \ldots, a_m)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_m)$ with non-increasing components and such that $\sum_{i=1}^m a_i = \sum_{i=1}^m b_i$ we write $\mathbf{a} \succ \mathbf{b}$ and say that vector $\mathbf{a}$ majorizes vector $\mathbf{b}$ (or vector $\mathbf{b}$ is majorized by vector $\mathbf{a}$) if*

$$\sum_{i=1}^{k} a_i \geq \sum_{i=1}^{k} b_i, \ 1 \leq k \leq m.$$

The monograph [9] gives a comprehensive exposition of numerous aspects of majorization theory and its applications. For our purposes, we formulate several relevant facts with respect to H_n^m, the set of all allocation vectors.

Definition 3. *A function $f : H_n^m \to \mathbb{R}$ is called Shur convex over H_n^m if for any allocation vectors $\mathbf{a}, \mathbf{b} \in H_n^m$ it follows from $\mathbf{a} \succ \mathbf{b}$ that $f(\mathbf{a}) \geq f(\mathbf{b})$.*

For a convex function $h : \mathbb{R} \to \mathbb{R}$, function $f(\mathbf{a}) = \sum_{i=1}^m h(a_i)$ is known to be Shur convex [9]. Applying this result, we immediately deduce the following statement.

Lemma 2. *For Problem $P|p_j = 1|\sum C_j$, an allocation vector that is majorized by any other allocation vector is an optimal allocation vector.*

It is fairly easy to see that an optimal allocation vector can be found by Algorithm SPT.

Lemma 3. *An allocation vector $\mathbf{y} = (y_1, y_2, \ldots, y_m)$ associated with a schedule found by Algorithm SPT is an optimal allocation vector.*

Proof. Assume that in Problem $P\,|p_j = 1|\sum C_j$ the number of jobs can be expressed as $n = mu + v$, where u and v are integers such $u \geq 1$ and $0 \leq v \leq m-1$.

If $v = 0$, then $C^* = u$ and Algorithm SPT assigns u jobs to each machine. In this case, the allocation vector is $(y_1, \ldots, y_m)$ with $y_i = u$, $1 \leq u \leq m$, and this is the only vector in which each component does not exceed C^*.

If $v \geq 1$, then Algorithm SPT assigns $u+1$ jobs to each of v machines (for convenience, assume that these are the first v machines $M_1, M_2, \ldots, M_v$ taken in the order of their numbering) and u jobs to each of the remaining $m - v$ machines. The corresponding allocation vector is $(y_1, \ldots, y_m)$ with $y_i = u+1$, $1 \leq u \leq v$, and $y_i = u$, $v+1 \leq u \leq m$. Clearly, in any allocation vector the number of components equal to $u+1$ cannot be less than v, since $(u+1)(v-1) + u(m-v+1) = mu + v - 1 = n-1$. On the other hand, any allocation vector with more than v components equal to $u+1$ will majorize the vector in which the number of such components is equal to v. Thus, the allocation vector generated by Algorithm SPT is either unique or is majorized by any other allocation vector. Therefore, such a vector is an optimal allocation vector. □

In fact, an optimal allocation vector can be found without running Algorithm SPT. Moreover, without running the algorithm, we may tell the position of any job in an optimal schedule of $n = mu + v$ UET jobs. Given a job J_j, $j = mu' + v'$, where $u' \leq u$ and $0 \leq v \leq m-1$, assign the job to be processed on machine M_m in the time interval $[u'-1, u']$, provided that $v' = 0$; otherwise, assign the job to be processed on machine $M_{v'}$ in the time interval $[u', u'+1]$.

4. UET Jobs under Resource Constraints

We now turn to Problem $P\,|p_j = 1, res \cdot 111|\sum C_j$. Recall that for this problem the set N of jobs is partitioned into q sets N^1, $N^2, \ldots, N^q$ such that each job of set N^r, and only of that set, requires resource r, $1 \leq r \leq q$. No two jobs of the same set N^r can be assigned to be processed in the same time slot. An input of this problem consists of q integers n_r, $1 \leq r \leq q$, where $n_r = |N^r| \geq 1$ is the number of jobs that require resource r. In this section, we assume that

$$n_1 \geq n_2 \geq \cdots \geq n_q; \tag{1}$$

otherwise, the corresponding renumbering of the resources can be done in $O(q \log q)$ time, which is polynomial with respect to the length of the input of the problem.

We may assume that $q > m$; otherwise, in an optimal schedule machine M_r processes the block of n_r UET jobs of set N^r in the time interval $[0, n_r]$, $1 \leq r \leq q$.

As shown in [7], Problem $P\,|p_j = 1, res \cdot 111|\sum C_j$ admits a polynomial-time algorithm based on its reduction to a min-cost max-flow problem. The network presented in [7] has $O(nq)$ vertices and $O(nqm)$ arcs (under the assumption that $q > m$). Thus, even if the fastest known min-cost max-flow algorithm by Orlin [11] is used, the approach from [7] cannot lead to an algorithm for solving Problem $P\,|p_j = 1, res \cdot 111|\sum C_j$ faster than in $O\left(n^2 q^2 m \log(nq)(m + (nq))\right)$ time.

Further in this section we describe a much more efficient algorithm. In fact, the most time consuming part of our algorithm is obtaining the numbering (1).

Simple as it is, our algorithm is based on several ingredients listed below.

Ingredient 1. It is proved in [7] that for Problem $P\,|res \cdot 111|\sum C_j$ with arbitrary processing times there exists an optimal schedule in which each machine is busy from time 0 and has no intermediate idle time. Similarly to Section 3, for

Problem $P\left|p_j = 1, res \cdot 111\right| \sum C_j$ with the UET jobs a vector $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ with non-increasing positive integer components is called a *feasible allocation vector* if there exists a feasible schedule S such that machine M_i completes its jobs at time $C^{[i]}(S) = x_i$, $1 \le i \le m$. An analogue of Lemma 2 holds, i.e., a feasible allocation vector $\mathbf{y}$ that is majorized by any feasible allocation vector $\mathbf{x}$ is an optimal allocation vector associated with an optimal schedule S^*. In what follows, we describe how find a feasible schedule with a given feasible allocation vector.

Ingredient 2. The second ingredient is the concept of a *composite* job. For Problem $P\left|res \cdot 111\right| \sum C_j$, associate each set N^r of jobs that require resource r, $1 \le r \le q$, with a composite job V_r, $1 \le r \le q$. The processing times of these composite jobs are defined by

$$p(V_r) = n_r,\ 1 \le r \le q.$$

We will call a schedule of the composite jobs a *group technology* schedule. Clearly, a group technology schedule can be converted into a feasible schedule for the original instance by replacing a composite job by the block of the corresponding original UET jobs.

Ingredient 3. Part of our algorithm relies on finding a preemptive group technology schedule for all or some of the composite jobs. In fact, what is needed is the following procedure. Given a (sub)set of the composite jobs $\{V_r, V_{r+1}, \ldots, V_q\}$, $m' = m - r + 1$ parallel machines $M_r, \ldots M_m$, and a vector $\mathbf{z} = (z_r, \ldots, z_m)$ of the desired completion times on these machines, we need to find a preemptive schedule of these composite jobs. In such a schedule, the last completion time on machine M_i must be equal to z_i, $r \le i \le m$. The processing of each composite job can be interrupted and resumed, possibly on a different machine, the parts of a preempted job to be scheduled without overlapping.

Such a schedule can be found by adapting the classical algorithm by McNaughton [10] for finding a preemptive schedule that minimizes the makespan. Below we present a modified version of McNaughton's algorithm, adapted for our purposes, as a generic procedure.

Procedure McN$(r, \mathbf{z})$

PARAMETERS: an integer r, $1 \le r \le m$ and a vector $\mathbf{z} = (z_r, \ldots, z_m)$ with non-increasing components.

INPUT: the composite jobs $V_r, V_{r+1}, \ldots V_q$; the parallel machines $M_r, \ldots M_m$.

CONDITIONS:

$$z_r - z_m \le 1,\ n_r < z_r,\ \sum_{k=r}^{q} n_k = \sum_{i=r}^{m} z_i. \tag{2}$$

OUTPUT: a feasible preemptive schedule for processing the given composite jobs, in which the last completion time on machine M_i is equal to z_i

```
Step1. : Create an artificial schedule $S_A$ in which the given composite jobs to be scheduled are placed on an auxiliary single machine, as a block starting at time 0.
Step 2.: Set $a := 0$. For each $i$ from $r$ to $m$ do the following:
    (a): Allocate the segment of schedule $S_A$ in the time interval $[a, a + z_i]$ to machine $M_i$ to be processed in the time interval $[0, z_i]$.
    (b): Update $a := a + z_i$.
Step 3.: Output the obtained schedule as schedule $S^{(r)}$.
```

Procedure McN$(r,\mathbf{z})$ takes $O(q-r)$ time, i.e., $O(q)$ time in the case of a complete set of composite jobs. The resulting schedule $S^{(r)}$ has at most $m'-1$ preemptions, since on each machine other than M_m at most one job is interrupted.

Lemma 4. *Under the conditions (2) Procedure McN$(r,\mathbf{z})$ outputs a required feasible schedule.*

Proof. The action of splitting schedule S_A into segments, as described in Step 2, guarantees that the last completion time on each machine M_i, $r \le i \le m$, is equal to z_i. The equality $\sum_{k=r}^{q} n_k = \sum_{i=r}^{m} z_i$ implies that the processing capabilities of the machines $M_r, \ldots M_m$ are equal to the processing requirements.

The only thing that should be demonstrated is the feasibility of schedule $S^{(r)}$, i.e., that the two segments of any preempted job are assigned to the intervals that do not overlap. Notice that the inequality $z_r - z_m \le 1$ implies that either all values of z_i are equal to z_r or some of them are equal to z_r while others are equal to $z_r - 1$. Thus, the inequality $n_r < z_r$ and the ordering (1) mean that the processing time of any composite job does not exceed any machine completion time z_i. Since in schedule $S^{(r)}$ any preempted job completes at time z_i on some machine M_i and starts at time 0 on machine M_{i+1}, $r \le i \le m-1$, it follows that no overlap occurs. $\square$

Ingredient 4. The final key component required for the design of our algorithm is the concept of the *genus* of an input.

Compute

$$(3)\qquad \begin{aligned} H_1 &:= n; \\ H_r &:= H_{r-1} - n_{r-1},\ 2 \le r \le q. \end{aligned}$$

Definition 4. *An instance of Problem $P\,|p_j = 1, res\cdot 111|\sum C_j$ is said to be of genus k, if k, $0 \le k \le m-1$, is the smallest integer such that*

$$(4)\qquad n_{k+1} < \lceil H_{k+1}/(m-k)\rceil .$$

For an instance of genus 0 of Problem $P\,|p_j = 1, res\cdot 111|\sum C_j$ the duration n_1 of the longest composite job V_1 is less that the optimal makespan $\lceil H_1/(m-0)\rceil = \lceil n/m\rceil$, computed for Problem $P\,|p_j = 1|\sum C_j$ of processing the jobs of the set $N^1 \cup \cdots \cup N^q$ on m machines $M_1, \ldots, M_m$, without taking into consideration the resource constraints. For an instance of genus $k > 0$, the duration of each of the k longest composite jobs V_i, $1 \le i \le k$, is no smaller than the optimal makespan computed for Problem $P\,|p_j = 1|\sum C_j$ of processing the jobs of the set $N^i \cup \cdots \cup N^q$ on machines $M_i, \ldots, M_m$, without taking into consideration the resource constraints. Notice that for an instance of genus $k > 0$, a subinstance of a given Problem $P\,|p_j = 1, res\cdot 111|\sum C_j$ related to the processing of the jobs of the set $N^{k+1} \cup \cdots \cup N^q$ on machines $M_{k+1}, \ldots, M_m$ is an instance of genus 0.

Example 1. For illustration, consider the following problem. There are m analysts who, as a team, have to update $q = 8$ files in a database. The number of records to be updated in a database file i is equal to n_i, $1 \le i \le 8$, and their values are given below as

$$\begin{aligned} n_1 &= 10,\ n_2 = 9,\ n_3 = 6,\ n_4 = 4, \\ n_5 &= 4,\ n_6 = 3,\ n_7 = 3,\ n_8 = 2. \end{aligned}$$

It can be assumed that updating any record takes the same time. Each database file cannot be accessed by several analysts simultaneously. The goal is to minimize the total (or average) completion time of all updates. Treating the m analysts as processing machines, records to be updated as UET jobs, and database files as $q = 8$ resources, we obtain Problem $Pm\,|p_j = 1, res\ 8111|\sum C_j$.

Thus, there are $n = H_1 = 41$ UET jobs. If there are 4 machines, i.e., $m = 4$, we have that $n_1 = 10 < \lceil H_1/m \rceil = \lceil 41/4 \rceil = 11$, i.e., (4) holds for $k = 0$ and this is a genus 0 instance.

If $m = 5$, then we compute $H_2 = 31$ and $H_3 = 22$. We see that $n_1 = 10 > \lceil H_1/m \rceil = \lceil 41/5 \rceil = 9$ and $n_2 = 9 > \lceil H_2/(m-1) \rceil = \lceil 31/4 \rceil = 8$. On the other hand, $n_3 = 6 < \lceil H_3/(m-2) \rceil = \lceil 22/3 \rceil = 8$, i.e., (4) holds for $k = 2$ and this is a genus 2 instance. Notice that the subinstance associated with the jobs of sets $N^3, \dots, N^8$ to be processed on machines M_3, M_4 and M_5 is a genus 0 instance.

We are now ready to present an algorithm that solves Problem $P\,|p_j = 1, res \cdot 111|\sum C_j$. The algorithm first determines the genus k of the given instance. Then the algorithm finds a schedule of the corresponding composite jobs. If $k = 0$ then Procedure McN$(1, \mathbf{z})$ is called, where $\mathbf{z}$ is the optimal allocation vector found without taking into consideration the resource constraints, as defined in Section 3. If $k > 1$, then a composite job V_i is assigned to be processed alone on machine M_i, $1 \le i \le k$. To schedule the remaining jobs, Procedure McN$(k+1, \mathbf{z})$ with an appropriate vector $\mathbf{z}$ is called. The obtained group technology schedule of the composite jobs is converted into an optimal schedule of the original UET jobs.

Algorithm 1

Step 1.: If necessary, renumber the resources so that (1) holds. Compute the values H_r, $1 \le r \le m-1$. Determine k, the genus of a the given instance of the problem. If $k > 0$, go to Step 2; otherwise, define $H' = n$, $m' = m$ and go to Step 3.

Step 2.: Create a (partial) schedule S' in which each machine M_i processes a single composite job V_i, $1 \le i \le k$. Define $H' = H_{k+1}$ and $m' = m - k$.

Step 3. : Define $u' = \lfloor H'/m' \rfloor$ and $v' = H' - u'm'$, i.e., represent H' as $H' = u'm' + v'$. Define a vector $\mathbf{z} = (z_{k+1}, \dots, z_m)$ with $m' = m - k$ components such that either all of them are equal to u' (provided that $v' = 0$) or the first v' of these components are equal to $u' + 1$, while the remaining components are equal to u'. Call Procedure McN$(k+1, \mathbf{z})$ and obtain a preemptive schedule $S^{(k+1)}$ of processing the composite jobs $V_{k+1}, \dots, V_q$ on machines $M_{k+1}, \dots, M_m$.

Step 4.: If $k > 0$, combine schedule S' on machines $M_1, \dots, M_k$ with schedule $S^{(k+1)}$ on machines $M_{k+1}, \dots, M_m$ into a complete preemptive group technology schedule S_{GP} of processing the composite jobs $V_1, \dots, V_q$ on machines $M_1, \dots, M_m$. If $k = 0$, then rename schedule $S^{(k+1)}$ as schedule S_{GP}. Convert schedule S_{GP} into schedule S^* of processing the original UET jobs by replacing any segment of a composite job V_r of length t by a block of t UET jobs of set N^r, $1 \le r \le q$.

The running time of Algorithm 1 is $O\,(n + q \log q)$. In fact, finding the numbering (1) requires $O\,(q \log q)$ time, Step 2 takes $O\,(q - k)$ time, and Step 4 takes $O\,(n)$ time. Below we prove the optimality of the algorithm.

Theorem 1. *Schedule S^* found by Algorithm 1 is optimal for Problem $P\,|p_j = 1, res\cdot 111|\sum C_j$.*

Proof. First, notice that S^* is a feasible schedule, i.e., no two jobs that require the same resource are assigned to the same time interval.

In terms of the group technology schedule S_{GP}, this means that the two pieces of any preempted composite job do not overlap. If $k > 1$, then the jobs in schedule S' are processed without preemption. Thus, for any genus k in order to prove the feasibility, we need to check the output of Procedure McN$(k+1, \mathbf{z})$. Such feasibility is guaranteed by Lemma 4, provided that the conditions (2) hold. To see this, observe that in the call of Procedure McN$(k+1, \mathbf{z})$ either all components of vector $\mathbf{z}$ are equal or differ by 1, as required by (2). For a genus k instance, we have that $n_{k+1} < \lceil H_{k+1}/(m-k)\rceil = \lceil H'/m'\rceil = z_{k+1}$, which is also part of (2). Finally, $\sum_{i=k+1}^{m} z_i = H'$, and on the other hand H' is the number of the original UET jobs in the set $N^{k+1}\cup\cdots\cup N^q$, equal to $\sum_{r=k+1}^{q} n_r$. Thus, all conditions (2) are satisfied, i.e., due to Lemma 4 schedule S_{GP} is a feasible group technology schedule, which in turn implies that S^* is a feasible schedule of the original UET jobs.

If $k = 0$, then for vector $\mathbf{z}$ found in Step 3, the vector $\mathbf{y} = (y_1, \ldots, y_m)$ that coincides with vector $\mathbf{z}$ is a feasible allocation vector associated with schedule S^*. If $k > 0$ then a feasible allocation vector is $\mathbf{y} = (y_1, \ldots, y_m)$, where

$$\begin{aligned} y_i &= n_i,\ 1 \le i \le k; \\ y_i &= z_i,\ k+1 \le i \le m. \end{aligned}$$

In any case, the value of the objective function of schedule S^* is equal to $\frac{1}{2}\sum_{r=1}^{q} y_i\,(y_{i+1})$. Similarly to Section 3, this function is Shur convex over the set of all feasible allocation vectors. To prove the optimality, we rely on Ingredient 1 above and show that vector $\mathbf{y}$ is majorized by any feasible allocation vector.

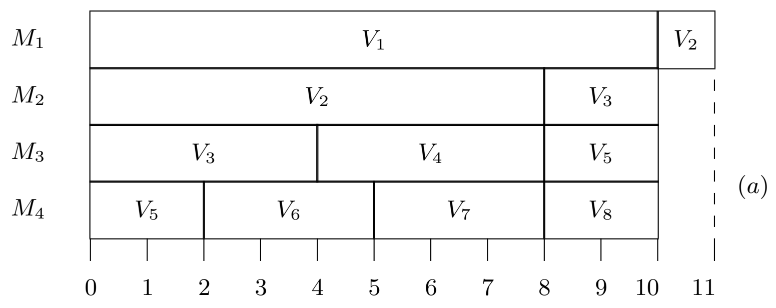
If vector $\mathbf{y}$ is not optimal, there would exist a feasible allocation vector $\mathbf{x}$ such that $\mathbf{y} \succ \mathbf{x}$.

Assume that $k > 0$ and show that if $\mathbf{y} \succ \mathbf{x}$ then $y_i = x_i,\ 1 \le i \le k$. Indeed, in any feasible schedule the processing of all jobs of set N^1 takes n_1 time units, i.e., there exists a machine which completes its jobs no earlier than time n_1. Since in a feasible schedule associated with the vector $\mathbf{x}$ the component x_1 corresponds to the latest completion time among all machines, it follows that $x_1 \ge n_1$. However, the assumption $\mathbf{y} \succ \mathbf{x}$ implies that $y_1 = n_1 \ge x_1 \ge n_1$ so that $y_1 = x_1$.

Further, $\mathbf{y} \succ \mathbf{x}$ implies that $y_1 + y_2 \ge x_1 + x_2$, which is equivalent to $y_2 = n_2 \ge x_2$. Again, the component x_2 corresponds to the latest completion time among all machines except machine M_1, so that $x_2 \ge n_2$, from which we immediately deduce that $y_2 = x_2$. Extending this argument, we conclude that if $\mathbf{y} \succ \mathbf{x}$ then $y_i = x_i,\ 1 \le i \le k$.

For any genus k, the components $y_i,\ k+1 \le i \le m$, are associated with a schedule for a (sub)instance of genus 0 of processing H' UET jobs on m' machines. These components are found exactly as done in Section 3, i.e., due to Lemma 3 these components serve as an optimal allocation vector for this (sub)instance, even if the resource constraints are ignored.

Thus, vector $\mathbf{y}$ cannot majorize another feasible allocation vector and is an optimal allocation vector. Therefore, S^* is an optimal schedule. $\square$



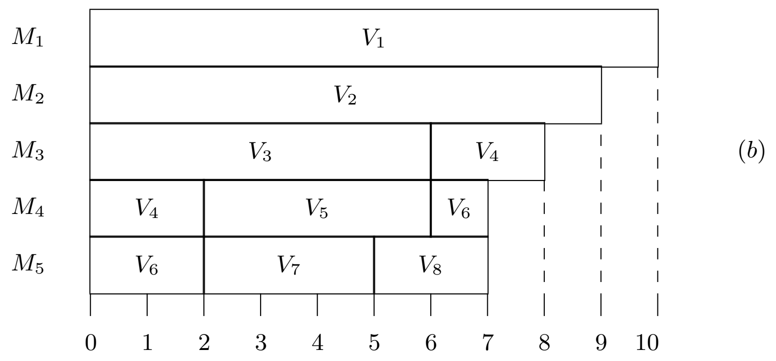


FIG. 1. Schedule S_{GP} for intance in Example 1: (a) $m = 4$; (b) $m = 5$

Example 2. To illustrate Algorithm 1, consider the instance in Example 1. In the case of $m = 4$ this is a genus 0 instance, and the corresponding group technology schedule S_{GP} is found as described in Step 3 by calling Procedure McN$(1, \mathbf{z})$, where $\mathbf{z} = (11, 10, 10, 10)$; see Figure 1(a).

If $m = 5$, we obtain a genus 2 instance, with composite jobs V_1 and V_2 assigned to machines M_1 and M_2, respectively, as described in Step 2. To schedule the remaining 6 composite jobs on the remaining 3 machines, Procedure McN$(3, \mathbf{z})$ is called, where $\mathbf{z} = (8, 7, 7)$. The resulting schedule is shown in Figure 1(b).

In either case, the resulting optimal schedule for 41 original UET jobs is obtained from schedule S_{GP} by replacing each piece of job V_r by the block of the jobs of set N^r. In the case of $m = 4$ the objective function is equal to $\frac{1}{2}(11 \cdot 12 + 3 \cdot 10 \cdot 11) = 231$, while for $m = 5$ the corresponding value is $\frac{1}{2}(10 \cdot 11 + 9 \cdot 10 + 8 \cdot 9 + 2 \cdot 7 \cdot 8) = 192$.

5. Conclusion

In this paper, we present an algorithm for minimizing the total completion time of the UET jobs on parallel machines with resource constraints. Our algorithm is much simpler and several orders faster that the previously known method [7].

For future research, to resolve the complexity status of Problem $P|res \cdot 111|\sum C_j$ with arbitrary processing times is of a considerable interest. It is likely that the problem is NP-hard for $m = 2$. If Problem $P|res \cdot 111|\sum C_j$ is indeed NP-hard, design of approximation algorithms would be a good research goal. An approximation algorithm for the problem described in [7] is left with a large gap between an upper bound and a lower bound on its worst case performance, and providing an algorithm with a tight worse case ratio is an interesting task.

References

[1] J. Błażewicz, N. Brauner, G. Finke, *Scheduling with discrete resource constraints*, In: Leung, J.Y.-T. (ed.), *Handbook of scheduling. Algorithms, models, and performance analysis*, Chapman & Hall/CRC, Boca Raton, 2004, 23-1–23-18.

[2] J. Błażewicz, J.K. Lenstra, A.H.G. Rinnooy Kan, *Scheduling subject to resource constraints: classification and complexity*, Discr. Appl. Math., **5** (1983), 11–24. Zbl 0516.68037

[3] J. Błażewicz, K.H. Ecker, E. Pesch, G. Schmidt, M. Sterna, J. Weglarz, *Scheduling under resource constraints*, In: J. Błażewicz et al. (eds.), *Handbook of Scheduling*, Springer, Cambridge, 2019, 475–525.

[4] R.W. Conway, W.L. Maxwell, L.W. Miller, *Theory of scheduling*, Addison-Wesley, Reading, 1967. Zbl 1058.90500

[5] E.B. Edis, C. Oguz, I. Ozkarahan, *Parallel machine scheduling with additional resources: notation, classification, models and solution methods*, Eur. J. Oper. Res., **230**:3 (2013), 449–463. Zbl 1317.90116

[6] E. Hebrard, M.-J. Hugueta, N. Jozefowiez, A. Maillard, C. Pralet, G. Verfaillie, *Approximation of the parallel machine scheduling problem with additional unit resources*, Discr. Appl. Math., **215** (2016), 126–135. Zbl 1356.90055

[7] T. Janssen, C. Swennenhuis, A. Bitar, T. Bosman, D. Gijswijt, L. van Iersel, S. Dausère-Pérèz, C. Yugma, *Parallel machine scheduling with a single resource per job*, 2018, arXiv:1809.05009v3.

[8] H. Kellerer, V.A. Strusevich, *Scheduling problems for parallel dedicated machines under multiple resource constraints*, Discr. Appl. Math., **133**:1-3 (2003), 45–68. Zbl 1053.90039

[9] A.W. Marshall, I. Olkin, *Inequalities: theory of majorization and its applications*, Academic Press, New York etc., 1979. Zbl 0437.26007

[10] R. McNaughton, *Scheduling with deadlines and loss functions*, Manag. Sci., **6** (1959), 1–12. Zbl 1047.90504

[11] J.B. Orlin, *A faster strongly polynomial minimum cost flow algorithm*, in *Proc. 20th ACM Symp. Theory Comput., STOC'88*, 1988, 377–387.

[12] V.A. Strusevich, *Approximation algorithms for makespan minimization on parallel machines under resource constraints*, J. Oper. Res. Soc., **72**:9 (2021), 2135–2146.

Vitaly Aleksandrovich Strusevich
Sherwood Road,
Welling, Kent,
DA16 2SJ, U.K.
Email address: sv02pri@gmail.com