S(e)MR ISSN 1813-3304

СИБИРСКИЕ ЭЛЕКТРОННЫЕ
МАТЕМАТИЧЕСКИЕ ИЗВЕСТИЯ

Siberian Electronic Mathematical Reports

http://semr.math.nsc.ru

Том 19, №2, стр. 586–600 (2022) УДК 519.8
DOI 10.33048/semi.2022.19.049 MSC 90B35

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

MINIMIZING MAKESPAN FOR PARALLELIZABLE JOBS WITH ENERGY CONSTRAINT

A. KONONOV, YU. ZAKHAROVA

Abstract. We investigate the problem of scheduling parallelizable jobs to minimize the makespan under the given energy budget. A parallelizable job can be run on an arbitrary number of processors with a job execution time that depends on the number of processors assigned to it. We consider malleable and moldable jobs. Processors can vary their speed to conserve energy using dynamic speed scaling. Polynomial time algorithms with approximation guarantees are proposed. In our algorithms, a lower bound on the makespan and processing times of jobs are calculated. Then numbers of utilized processors are assigned for jobs and a feasible solution is constructed using a list-type scheduling rule.

Keywords: parallelizable job, speed scaling, scheduling, approximation algorithm.

1. Introduction

We investigate the problem of scheduling a set of jobs $\mathcal{J} = \{1, \dots, n\}$ on m speed scalable parallel processors. Each job $j \in \mathcal{J}$ is characterized by processing volume (work) W_j and the upper bound $\delta_j \le m$ on the possible number of utilized processors. We consider malleable and moldable jobs [2]. In the case of moldable jobs, the number of utilized processors is chosen by the scheduler before starting

Kononov, A., Zakharova, Yu., Minimizing makespan for parallelizable jobs with energy constraint.

The work is supported by Russian Science Foundation (grant 21-41-09017).
Received May, 13, 2022, published August, 30, 2022.

the job, and it is not changed until the job termination. In the case of malleable jobs, we may change the number of processors utilized by a job during the schedule. We investigate the linear case, when the runtime of a job decreases linearly with the number of processors assigned to it, and W_j is the execution time of job j on one processor with unit speed. We assume that all jobs arrive at time step 0. Job preemption, migration, and precedence constraint might or might not be allowed in the exploring of scheduling in this paper.

The standard homogeneous model in speed-scaling is considered. When a processor runs at a speed s, then the rate with which the energy is consumed (the *power*) is s^α, where $\alpha > 1$ is a constant. Each of m processors may operate at variable speed. However, we assume that the total work W_j of a job $j \in \mathcal{J}$ should be uniformly divided between the utilized processors, i.e. if job j uses m_j processors, then processing volumes are the same for all m_j processors, and all these processors run at the same speed. It is supposed that a continuous spectrum of processor speeds is available.

The aim is to find a feasible schedule with the smallest value of the maximum completion time (makespan, $C_{\max}$) so that the energy consumption is not greater than a given energy budget E. This is a natural assumption in the case when the energy of a battery is fixed, i.e. the problem finds applications in computer devices whose lifetime depends on a limited battery efficiency (for example, multi-core laptops). Moreover, the bicriteria problems of minimizing energy consumption and a scheduling metric arise in real practice. The most obvious approach is to bound one of the objective functions and optimize the other. The energy of the battery can be reasonably estimated, so we bound the energy used, and optimize the regular timing criterion.

The moldable and malleable variants of the speed-scaling scheduling subject to bound on energy consumption are denoted by $P|any, \delta_j, energy|C_{\max}$ and $P|var, \delta_j, energy|C_{\max}$, respectively.

2. Previous Research

Now we review the known results for the classical makespan minimization problem of scheduling moldable and malleable jobs with given durations and without energy constraint. The non-preemptive problem with moldable jobs is NP-hard. For problems $P2|any|C_{\max}$ and $P3|any|C_{\max}$ pseudopolynomial algorithms based on dynamic programming have been proposed in [4]. A PTAS is known for $Pm|any|C_{\max}$ [6]. List-type approximation algorithms were developed for problem $P|any, \delta_j, prec|C_{\max}$, scheduling moldable jobs with linear speedup, limits on parallelism δ_j and precedence constraints [14]. Algorithms use Earliest Completion Time strategy and take into account alternative starting times and available numbers of processors.

Constant factor approximation algorithms have been obtained for the more general problems with nondecreasing sublinear speedup functions of jobs on the number of utilized processors (NdSub). A $\frac{2}{1-\frac{1}{m}}$-approximation algorithm for problem $P|any, NdSub, \delta_j|C_{\max}$ initially considers all jobs as single-processor ones and builds a schedule using LPT (Largest Processing Time First) rule. Then this algorithm iteratively increases the number of utilized processors for long jobs [1]. Similar approaches were used in [5, 9]. For the instances with precedence constraints, a two-stage approximation algorithm is known [11]. At the first stage, processor allotments are selected by a 2-approximation algorithm, taking into account the

Table. 1. Approximation results for various types of jobs and makespan criterion

Type of Jobs	Preemptive and Migrative	Non-preemptive	Precedence Constraints
Rigid	OPT$+\varepsilon$ [8]	$2-\frac{1}{m}$ [8]	$\frac{2-q}{1-q}$ [8]
Moldable	?	$2-\frac{2}{m+1}$	$\frac{3+\sqrt{5}}{2}$
Malleable	poly	–	?

longest path in the partial order graph and the total load of processors. At the second stage, a feasible schedule of jobs with given sizes is constructed using a precedence-dependent list-scheduling rule.

The problem with malleable independent jobs and linear speedup is solvable in linear time [3]. The problem with precedence constraints as chains is polynomially solvable when the number of processors is fixed, and it is NP-hard when the number of processors is a part of the input [3].

In our previous research, we investigated the speed-scaling problem with rigid and single mode jobs [7, 8]. A rigid job can be processed on any subset of parallel processors of the given size. A single mode job uses the prespecified subset of dedicated processors. NP-hardness proofs and approximation algorithms were proposed. Our strategies are based on relaxed formulations of the problems as convex programs solved by the Ellipsoid method and the KKT-conditions. These programs allow us to obtain lower bounds on the makespan and auxiliary processing times of jobs. Then we find feasible schedules using various greedy rules.

Our results. We develop two-stage algorithms for the following cases:

- malleable jobs,
- moldable jobs,
- moldable jobs with precedence constraints.

At the first stage, a lower bound on the makespan and processing times of jobs are calculated. Then, at the second stage, numbers of utilized processors are assigned for jobs, and a feasible solution is constructed by a list-type scheduling rule. Constant factor approximation guarantees are provided.

In Table 1 we present the previously obtained results, new results provided in this paper and open questions (marked by "?") for various types of jobs.

3. Malleable Jobs

In this section we consider malleable jobs, that can be executed by more than one processor in parallel, decreasing in this way its total execution time. Preemption, migration and changing the number of used processors are allowed. The duration of job j on k processors is supposed to be $\frac{p_j}{k}$, when p_j is the processing time of the job on one processor.

The considered speed-scaling problem with malleable jobs is polynomially solvable. Indeed, the lower bound on the objective function is calculated by solving the following convex program:

$$LB \to \min, \tag{1}$$

$$\frac{1}{m}\sum_{j=1}^{n} p_j \le LB, \tag{2}$$

$$\frac{p_j}{\delta_j} \le LB,\ j \in \mathcal{J}, \tag{3}$$

$$\sum_{j\in\mathcal{J}} p_j \left(\frac{W_j}{p_j}\right)^{\alpha} \le E, \tag{4}$$

$$p_j \ge 0,\ j \in \mathcal{J}. \tag{5}$$

Here p_j denote the processing time of job j on one processor. Inequality (2) states that the total load of all processors is no more than $LB \cdot m$. Constraints (3) guarantee that the duration of any job j does not exceed LB, when the maximum possible number δ_j of processors is used. Inequality (4) ensures that the total energy consumption is not greater than the budget E. Program (1)-(5) can be solved in polynomial time using Karush-Kuhn-Tucker conditions [10] similar to [8] (see description in Appendix).

We construct the optimal schedule using the obtained processing times of jobs and the algorithm of McNaughton [12]: Jobs are assigned consecutively, one after another, to the processors starting from the first one. If the interval of processing some job on some processor i exceeds LB, then the job is divided into parts:

- the first part finishing by LB on processor i;
- several parts which are executed during interval $[0, LB)$ on processors $i+1, \dots, i+l-1$;
- the last part starting at time 0 on processor $i+l$.

This procedure is repeated for all jobs. The time complexity is $O(n)$.

Theorem 1. *An optimal schedule can be found in polynomial time for scheduling problem* $P|var, \delta_j, energy|C_{\max}$.

We note that an optimal schedule for malleable jobs with $\delta_j = m,\ j \in \mathcal{J}$, can be easily found in linear time by assigning m processors to each job and arbitrary ordering the jobs. In this case, the processing times of jobs are $p_j = \frac{W_j}{m}\left(\frac{E}{\sum_{i\in\mathcal{J}} W_i}\right)^{1/(1-\alpha)}$ and all processors work with the same constant speed.

4. Moldable Jobs

In the case of moldable jobs, the number of required processors is chosen by the scheduler before starting the job, and it is not changed until the job termination. Suppose that a moldable job j can be executed on up to $\delta_j \le m$ processors. Preemption and migration are not allowed. The problem is NP-hard even in the case of $\delta_j = 1$ for all $j \in \mathcal{J}$ and $m = 2$ (the well known Partition problem is reduced to it). So, here we develop an approximation algorithm.

The lower bound LB on the makespan and processing times of jobs p_j can be found in polynomial time using model (1)–(5) for malleable jobs. Then, if we assign the number of utilized processors $m_j := \delta_j$ for all jobs $j \in \mathcal{J}$, and construct a feasible schedule by the "non-preemptive list-scheduling" algorithm proposed in [8], then we obtain a $\left(2 - \frac{1}{m}\right)$-approximate solution of the problem as in the case of rigid jobs.

Now we propose an algorithm with approximation ratio of $\left(2 - \frac{2}{m+1}\right)$ (see Algorithm 1). For example, we have $\frac{3}{2}$-approximation algorithm for rigid jobs and $\frac{4}{3}$-approximation algorithm for moldable jobs in the case of two processors.

In Algorithm 1, all jobs are initially considered as single-processor ones, and a feasible schedule is constructed using the Longest Processing Time First strategy. Then, at each iteration, we identify a job, that defines the schedule length, and increase the number of utilized processors for it until an approximate solution with the stated guarantee factor will be obtained. The running time of each iteration is $O(n\log m)$ as a subset of sequential processors should be assigned for multiprocessor jobs, and each single-processor job should be placed on the specific processor. The total time complexity of Algorithm 1 is equal to $O(n^2\log m)$.

Recall that $LB \geq \max\{\mathcal{A}; \max\limits_{j\in\mathcal{J}} \frac{p_j}{\delta_j}\}$, where $\mathcal{A} := \frac{1}{m}\sum\limits_{j\in\mathcal{J}} p_j$. We will show that the presented algorithm has $\left(2 - \frac{2}{m+1}\right)$ approximation ratio. Initially, we provide properties of the constructed schedule and then prove the main result of this section.

Algorithm 1 List Schedule for Moldable Jobs

```
1: Sort the jobs in non-increasing order based on execution times $p_j$. The jobs are assigned to processors such that the next job is placed on the processor with the current minimum finish time. Put the number of utilized processors $m_j = 1$ for all $j \in \mathcal{J}$. Denote the obtained schedule by $S$ and its length by $C_{\max}(S)$.
2: Find a job $i = \operatorname{argmax}_{j\in\mathcal{J}} \frac{p_j}{m_j}$ and set $h = \max_{j\in\mathcal{J}} \frac{p_j}{m_j}$.
3: If either $C_{\max}(S) \neq h$, or $m_i > 1$, or $\delta_i = 1$, then go to Step 5. If $C_{\max}(S) = h$, but $h \leq \left(2 - \frac{2}{m+1}\right) \cdot LB$, then also go to Step 5.
4: Set $m_i := \left\lceil \frac{p_i}{\left(2-\frac{2}{m+1}\right)\cdot LB} \right\rceil$. Construct a schedule $S$ in the following form. The jobs $j \in \mathcal{J}$ with $m_j > 1$ are placed on the required number of processors at starting time 0 (the total number of utilized processors will be no more than $m$). We schedule the remaining jobs in the same way as at Step 1. If $C_{\max}(S') > h$, then put $m_i := 1$ and go to Step 5, else set $S := S'$ and go to Step 2.
5: Return schedule $S$.
```

Lemma 1. *Let $m \geq 2$ and S be the schedule obtained on some Step of Algorithm 1. Then $\frac{p_i}{m_i} > \left(1 - \frac{1}{m+1}\right) \cdot LB$ for all $i \in \mathcal{J}$ such that $m_i > 1$.*

Proof. Assume that $\frac{p_i}{m_i} \leq \left(1 - \frac{1}{m+1}\right) \cdot LB$ for some job i utilizing more than one processor, i.e. $p_i \leq \left(1 - \frac{1}{m+1}\right) \cdot LB \cdot m_i$. Then we see $\frac{p_i}{m_i-1} \leq \frac{\left(1-\frac{1}{m+1}\right)\cdot LB\cdot m_i}{m_i-1} \leq 2\left(1 - \frac{1}{m+1}\right) \cdot LB = \left(2 - \frac{2}{m+1}\right) \cdot LB$. This is contrary to the choice of m_i in Step 4. □

Lemma 2. *At Step 4 of Algorithm 1 we have* $\sum\limits_{j\in\mathcal{J}:\ m_j>1} m_j \le m$ *after assigning* $m_i := \left\lceil \frac{p_i}{\left(2-\frac{2}{m+1}\right)\cdot LB}\right\rceil$.

Proof. Assume that $\sum\limits_{j:\ m_j>1} m_j \ge m+1$, then from Lemma 1 the total load of processors by jobs with $m_j > 1$ will be greater than

$$(m+1)\left(1-\frac{1}{m+1}\right)\cdot LB = m\cdot LB \ge \sum_{j\in\mathcal{J}} p_j.$$

This leads to a contradiction. □

Lemma 3. *Let S be the schedule obtained on Step 1 or 4 of Algorithm 1, and y is the execution time of the first job that starts at time $t > 0$ (if there is no such job, $y = 0$). Then $C_{\max}(S) \le \max\{h(S); \mathcal{A} + y\left(1-\frac{1}{m}\right)\}$, where $h(S) = \max_{j\in\mathcal{J}} \frac{p_j}{m_j}$ is the execution time of the largest job in schedule S.*

Proof. If $C_{\max}(S)$ corresponds to the completion time of a job scheduled at time 0, then $C_{\max}(S) = h(S)$. Otherwise, $C_{\max}(S)$ is equal to the finish time of a job i, which was scheduled later (see Fig. 1). It is easy to see that $m_i = 1$ and $p_i \le y$. Job i starts at time $t' = (C_{\max}(S) - p_i)$. From the construction of S in Algorithm 1, none of the processors has completed before time t'. Set $\mathcal{A}' := \frac{1}{m}\sum_{j\in\mathcal{J}'} p_j$, where $\mathcal{J}'$ is the subset of the jobs already started before time t'. Then $t' \le \mathcal{A}' \le \mathcal{A} - \frac{p_i}{m}$. As a result we have

$$C_{\max}(S) - p_i \le \mathcal{A} - \frac{p_i}{m}$$

and

$$C_{\max}(S) \le \mathcal{A} + p_i\left(1-\frac{1}{m}\right) \le \mathcal{A} + y\left(1-\frac{1}{m}\right).$$

□

Lemma 4. *Let S be the schedule obtained on Step 1 or 4 of Algorithm 1, and y is the execution time of the first job that starts at time $t > 0$ (if there is no such job, $y = 0$). Then*
1. *$y \le \frac{p_i}{m_i}$ for all $i \in \mathcal{J}$ such that $m_i > 1$;* **2.** *$C_{\max}(S) \le \max\{h(S); \frac{2\mathcal{A}}{\left(1+\frac{1}{m}\right)}\}$.*

Proof. **1.** From Lemma 1 we have $\frac{p_i}{m_i} > \left(1-\frac{1}{m+1}\right)\cdot LB$ for all jobs $i \in \mathcal{J}$ using more than one processor. Assume that $y \ge \left(1-\frac{1}{m+1}\right)\cdot LB$, then all single-processor jobs starting at time 0 have durations $p_j \ge \left(1-\frac{1}{m+1}\right)\cdot LB$. So the total load of m processors by jobs starting at zero time and by the job corresponding to y will be greater than

$$(m+1)\left(1-\frac{1}{m+1}\right)\cdot LB = m\cdot LB \ge \sum_{j\in\mathcal{J}} p_j.$$

This leads to a contradiction. Therefore, $y < \left(1-\frac{1}{m+1}\right)\cdot LB$ and $y < \frac{p_i}{m_i}$ for all $i \in \mathcal{J}$ such that $m_i > 1$.
2. The execution time of the job corresponding to y is no more than the durations of jobs, starting at time 0. So, we have $\mathcal{A} \ge m\frac{y}{m} + \frac{y}{m}$. In other words, $y \le$

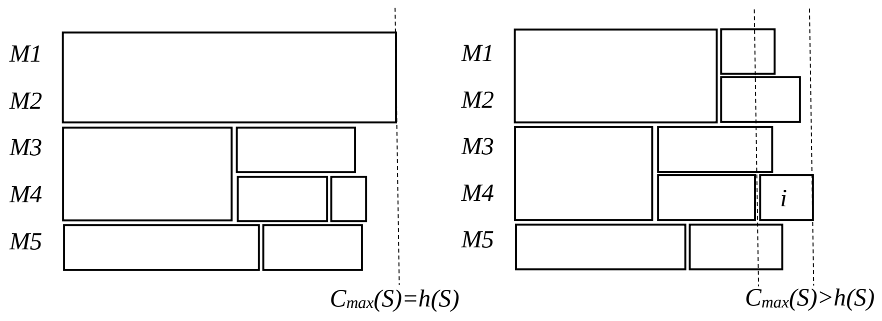

FIG. 1. Types of schedules constructed by Algorithm 1.

$\frac{\mathcal{A}}{(1+\frac{1}{m})}$. Substituting this expression in the result of Lemma 3, we obtain $C_{\max}(S) \leq \max\left\{h(S); \frac{2\mathcal{A}}{(1+\frac{1}{m})}\right\}$. $\square$

Theorem 2. *Let S be the schedule obtained on Step 5 of Algorithm 1. Then* $C_{\max}(S) \leq \left(2 - \frac{2}{m+1}\right) \cdot LB$.

Proof. There are two cases (see Fig. 1).
Case 1. $C_{\max}(S) \neq h(S) := \max_{j\in\mathcal{J}} \frac{p_j}{m_j}$. It is easy to check that $C_{\max}(S) \geq \max\{h(S);\mathcal{A}\}$. So, $C_{\max}(S) > h(S)$, and from Lemma 4 we have $C_{\max}(S) \leq \left(2 - \frac{2}{m+1}\right)\mathcal{A} \leq \left(2 - \frac{2}{m+1}\right) \cdot LB$.
Case 2. $C_{\max}(S) = h(S)$. We consider the various possible cases of termination in Algorithm 1.

Algorithm 1 terminated at Step 3 since
1) $m_i > 1$, so, due to the selection rule of m_i value we have $h(S) = \frac{p_i}{m_i} \leq \left(2 - \frac{2}{m+1}\right) \cdot LB$;
2) $\delta_i = 1$, therefore, $p_i = \frac{p_i}{\delta_i} \leq LB$;
3) $h(S) \leq \left(2 - \frac{2}{m+1}\right) \cdot LB$, thus, $C_{\max}(S) \leq \left(2 - \frac{2}{m+1}\right) \cdot LB$.

Algorithm 1 terminated at Step 4 as the constructed schedule S' has $C_{\max}(S') > h(S)$. Similar to the Proof of Lemma 4 we can show

$$C_{\max}(S') \leq \max\left\{h(S'); \frac{2\mathcal{A}}{(1+\frac{1}{m})}\right\}.$$

From our assumption $C_{\max}(S') > h(S) \geq h(S')$, so, $C_{\max}(S') \leq \frac{2\mathcal{A}}{(1+\frac{1}{m})}$. However, $C_{\max}(S) = h(S) < C_{\max}(S') \leq \frac{2\mathcal{A}}{(1+\frac{1}{m})} \leq \left(2 - \frac{2}{m+1}\right) \cdot LB$. $\square$

Theorem 3. *A* $\left(2 - \frac{2}{m+1}\right)$*-approximate schedule can be found in polynomial time for problem* $P|any,\delta_j,energy|C_{\max}$.

5. Moldable Jobs with Precedence Constraints

In this section, we consider the case of non-preemptive moldable jobs and precedence relations between jobs. If job j precedes job j' (we write $j \prec j'$), then j' cannot start until j is completed. The precedence constraints are represented by a directed acyclic graph $G = (J, A)$, where arc (j, j') belongs to set A if and only if job j must precede job j'.

We formulate the following convex problem to obtain a lower bound on the optimal makespan:

$$T \to \min, \tag{6}$$

$$\frac{1}{m}\sum_{j\in\mathcal{J}} p_j \le T, \tag{7}$$

$$C_j \le T,\ j \in \mathcal{J}, \tag{8}$$

$$\frac{p_j}{\delta_j} \le C_j,\ j \in \mathcal{J}, \tag{9}$$

$$C_j + \frac{p_{j'}}{\delta_{j'}} \le C_{j'},\ (j, j') \in A, \tag{10}$$

$$\sum_{j\in\mathcal{J}} W_j^{\alpha} p_j^{1-\alpha} \le E, \tag{11}$$

$$C_j \ge 0,\ p_j \ge 0,\ j \in \mathcal{J}. \tag{12}$$

Here variable p_j (C_j) represents the processing time of job $j \in \mathcal{J}$ on one processor (the estimation of the completion time of j). Inequalities (7), (8) and (9), (11) have the same sense as inequalities (2), (3), (4) in the model for the problem without precedence constraints (see Section 3). Inequality (10) allow estimating the completion times of jobs taking into account the precedence constraints. Convex program (6)-(12) implies a polynomial-time algorithm for calculating the lower bound, as convex programs can be solved to arbitrary precision by the Ellipsoid algorithm (see, e.g., [13]).

After solving program (6)-(12) we can identify the longest path in graph $G^\delta = (J, A, P^\delta)$ with lengths of job-vertices equal to $\frac{p_j}{\delta_j}$, i.e. $P^\delta = \left(\frac{p_1}{\delta_1}, \dots, \frac{p_n}{\delta_n}\right)$. We denote the length of this path by L^δ, and the average duration of jobs by $\mathcal{A} := \frac{1}{m}\sum_{j\in\mathcal{J}} p_j$. Then, the optimal makespan $C^*_{\max} \ge \max\{L^\delta;\ \mathcal{A}\}$.

Let $\mu \le \frac{m+1}{2}$ denote the maximum number of processors, that can be used by any job in the schedule constructed by Algorithm 2. Here the number of processors are assigned for each job at the first step, and the jobs are scheduled in accordance with the precedence constraints in $O(n^3)$ time at the second step.

Algorithm 2 List Schedule for Moldable Jobs with Precedence Constraints

```
1: Assign the number of processors for job $j \in \mathcal{J}$ equal to $m_j := \min\{\delta_j, \mu\}$.
   This defines the execution time $\bar{p}_j = \frac{p_j}{m_j}$ of job $j \in \mathcal{J}$ on each of the utilized
   processors.
2: Repeat the following procedure until all jobs have been scheduled:
   Let $\bar{\mathcal{J}}$ be the subset of unscheduled jobs whose predecessors all have already
   been scheduled. For each job from $\bar{\mathcal{J}}$ calculate the earliest possible starting
   time. Add the job with the smallest earliest starting time in the schedule.
```

We will show that Algorithm 2 has $\max\left\{\frac{m}{\mu}; \frac{2m-\mu}{m-\mu+1}\right\}$ approximation ratio. Initially we establish properties of the constructed schedule, and then provide constant factor approximation guarantee. Let $C^{apx}_{\max}$ denote the length of the schedule, constructed by Algorithm 2. Then, $C^{apx}_{\max} \geq \max\{L^{apx};\ \mathcal{A}\}$, where L^{apx} is the length of the longest path in graph $G^{apx} = (J, A, P^{apx})$ with the vector of job lengths equal to $P^{apx} = \left(\frac{p_1}{m_1}, \dots, \frac{p_n}{m_n}\right)$.

The constructed schedule contains three type of intervals $I_1,\ I_2,\ I_3$ (see Fig. 2), such that

- I_1 is the set of intervals, where at most $\mu - 1$ processors are used;
- I_2 is the set of intervals, where at least μ, but no more than $m-\mu$ processors are used;
- I_3 is the set of intervals, where at least $m - \mu + 1$ processors are used.

Therefore,

$$C^{apx}_{\max} = |I_1| + |I_2| + |I_3|, \tag{13}$$

$$m\mathcal{A} = \sum_{j \in \mathcal{J}} p_j \geq |I_1| + \mu|I_2| + (m - \mu + 1)|I_3|. \tag{14}$$

If we multiply (13) by $(m - \mu + 1)$ and subtract (14), then obtain

$$(m - \mu + 1)C^{apx}_{\max} \leq m\mathcal{A} + (m - \mu)|I_1| + (m - 2\mu + 1)|I_2|. \tag{15}$$

Now we provide the relation between the length L^δ of the critical path in G^δ and total durations of intervals from $I_1,\ I_2$.

Lemma 5. *Inequality $|I_1| + \frac{\mu}{m}|I_2| \leq L^\delta$ holds for the schedule, constructed by Algorithm 2.*

Proof. We find a critical chain of jobs in the schedule constructed by Algorithm 2. Let j_1 be any job that completes at time $C^{apx}_{\max}$. After we have obtained jobs $j_i \to j_{i-1} \to \dots \to j_1$ we find the next job j_{i+1} as follows:
Consider the latest time interval I in $I_1 \cup I_2$ that lies before the starting time of j_i. Identify the set $\mathcal{J}'$ of jobs that includes j_i and all its predecessor jobs that start after interval I. Since during interval I at most $(m - \mu)$ processors are busy, and since Algorithm 2 assigns at most μ processors to each job, all jobs from $\mathcal{J}'$ cannot be performed during interval I due to some predecessor jobs. As the next job j_{i+1}, we select any predecessor of j_i that is executed in interval I and completed at the end of I or later. This procedure terminates when the critical chain contains a job that starts before all intervals in $I_1 \cup I_2$. By the construction, the jobs of the critical chain cover all intervals in $I_1 \cup I_2$.

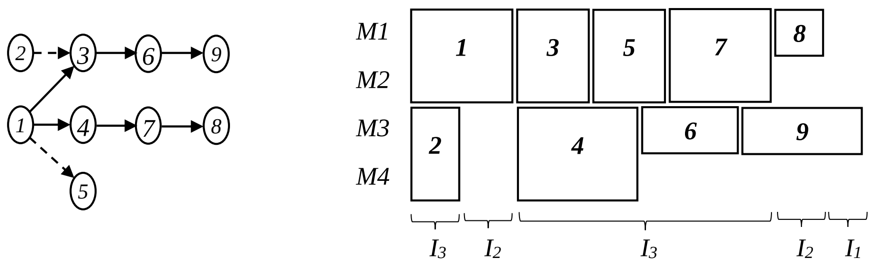

FIG. 2. Schedule S constructed by Algorithm 2, $L^\delta = 13$ (arcs of the longest paths are presented by solid lines), $C_{\max} = 18$. Here $n = 9$, $m = 4$, $\mu = 2$, $p = (8,4,6,10,6,4,8,2,5)$, $\delta = (4,2,3,2,3,1,2,1,1)$, $P^\delta = (2,2,2,5,2,4,4,2,5)$, $P^{apx} = (4,2,3,5,3,4,4,2,5)$.

Now we consider any job j of the obtained critical chain. If $m_j = \delta_j \le \mu$, then this job has the same length in the schedule and in the graph G^δ from the lower bound. The job can be executed in I_1 and I_2. If $m_j = \mu < \delta_j$, then this job has duration $\bar{p}_j = \frac{p_j}{\mu}$ in the schedule and length $\frac{p_j}{\delta_j}$ in graph G^δ. So, $\frac{p_j}{\delta_j} \ge \frac{\mu}{m}\bar{p}_j$ and the job can be executed only in I_2. The lower bound on the length of the critical chain in terms of graph G^δ is $|I_1| + \frac{\mu}{m}|I_2|$. At the same time this lower bound $|I_1| + \frac{\mu}{m}|I_2| \le L^\delta$ as L^δ is the length of the longest path in G^δ. $\square$

Lemma 6. *Let* $\frac{m}{\mu} \le \frac{2m-\mu}{m-\mu+1}$. *Then* $C^{apx}_{\max} \le \frac{2m-\mu}{m-\mu+1} C^*_{\max}$.

Proof. From the assumed inequality we have $(m-\mu+1) \le \frac{\mu}{m}(2m-\mu) = \frac{\mu}{m}(m-\mu)+\mu$, therefore, $(m-2\mu+1) \le \frac{\mu}{m}(m-\mu)$. Then, using (15) and Lemma 5 we obtain

$$(m-\mu+1)C^{apx}_{\max} \le m\mathcal{A} + (m-\mu)|I_1| + (m-2\mu+1)|I_2| \le$$
$$m\mathcal{A} + (m-\mu)|I_1| + \frac{\mu}{m}(m-\mu)|I_2| \le m\mathcal{A} + (m-\mu)L^\delta.$$

As $\max\{\mathcal{A};\ L^\delta\} \le C^*_{\max}$, the required inequality takes place:

$$(m-\mu+1)C^{apx}_{\max} \le mC^*_{\max} + (m-\mu)C^*_{\max} = (2m-\mu)C^*_{\max}.$$

$\square$

Lemma 7. *Let* $\frac{m}{\mu} \ge \frac{2m-\mu}{m-\mu+1}$. *Then* $C^{apx}_{\max} \le \frac{m}{\mu} C^*_{\max}$.

Proof. The assumed inequality is equivalent to $(m-\mu) \le \frac{m}{\mu}(m-2\mu+1)$. Hence, from (15) and Lemma 5 we have

$$(m-\mu+1)C^{apx}_{\max} \le m\mathcal{A} + (m-\mu)|I_1| + (m-2\mu+1)|I_2| \le$$
$$m\mathcal{A} + \frac{m}{\mu}(m-2\mu+1)|I_1| + (m-2\mu+1)|I_2| \le m\mathcal{A} + \frac{m}{\mu}(m-2\mu+1)L^\delta.$$

Due to $\max\{\mathcal{A};\ L^\delta\} \le C^*_{\max}$, we obtain:

$$(m-\mu+1)C^{apx}_{\max} \le mC^*_{\max} + \frac{m}{\mu}(m-2\mu+1)C^*_{\max} = \frac{m}{\mu}(m-\mu+1)C^*_{\max}.$$

□

Lemma 8. *For each $m \ge 2$ we have*

$$\min_{1\le\mu\le\frac{m+1}{2}} \max\left\{\frac{m}{\mu};\frac{2m-\mu}{m-\mu+1}\right\} < \frac{3+\sqrt{5}}{2} \approx 2.62.$$

The minimum is reached either on the integer above or on the integer below $\frac{1}{2}(3m-\sqrt{5m^2-4m})$.

Proof. Function $f_1(\mu) = \frac{m}{\mu}$ is decreasing and function $f_2(\mu) = \frac{2m-\mu}{m-\mu+1}$ is increasing. Therefore, the equality $f_1(\mu) = f_2(\mu)$ gives the minimum on the fractional value $\tilde{\mu} = \frac{1}{2}(3m-\sqrt{5m^2-4m})$ (it is the root of equation $\mu^2 - 3m\mu + m^2 + m = 0$). In this case we have

$$\frac{m}{\tilde{\mu}} = \frac{m}{(3m-\sqrt{5m^2-4m})/2} = \frac{m(3m+\sqrt{5m^2-4m})}{(2m^2+2m)} < \frac{3m+\sqrt{5}m}{2m} = \frac{3+\sqrt{5}}{2}.$$

Note that $f_1(\mu) = \frac{3+\sqrt{5}}{2}$ for $\mu = \frac{3-\sqrt{5}}{2}m$ and $f_2(\mu) = \frac{3+\sqrt{5}}{2}$ for $\mu = \frac{3-\sqrt{5}}{2}m + \frac{\sqrt{5}+1}{2}$. The length of interval $\left[\frac{3-\sqrt{5}}{2}m,\ \frac{3-\sqrt{5}}{2}m + \frac{\sqrt{5}+1}{2}\right]$ is greater than one and this interval contains value $\tilde{\mu}$. So, the minimum of $\max\left\{\frac{m}{\mu};\frac{2m-\mu}{m-\mu+1}\right\} < \frac{3+\sqrt{5}}{2}$ and it is reached either on the integer above or on the integer below $\tilde{\mu}$ for integer values of μ. □

As a result, we obtain the following theorem.

Theorem 4. *A $\left(\frac{3+\sqrt{5}}{2}\right)$-approximate schedule can be found in polynomial time for problem $P|any,\delta_j,prec,energy|C_{\max}$.*

Consider the case when $\delta_j \le qm$ for all $j \in \mathcal{J}$, $0 < q < 1$. If we assign the number of utilized processors $m_j = \delta_j$ for all jobs, and construct a feasible schedule by the "precedence-dependent list-scheduling" algorithm from [8], then we obtain a $\left(\frac{2-q}{1-q}\right)$-approximate solution of the problem as for rigid jobs. It is easy to see that $\frac{2-q}{1-q} > \frac{3+\sqrt{5}}{2}$ for all $q > \frac{\sqrt{5}-1}{\sqrt{5}+1} \approx 0.38$.

The complexity status of the problem with precedence dependent jobs of malleable type is open.

Conclusion

We provide approximation and exact algorithms for the speed scaling problems with the makespan criterion under the given energy budget. In our algorithms, initially, a lower bound on the makespan and processing times of jobs are calculated, and then a feasible solution is constructed using a list-type scheduling rule. In contrast to the previous research of speed scaling scheduling, we analyze parallelizable jobs, which can be run on an arbitrary number of processors up to the given upper level.

Appendix

Appendix contains a polynomial time algorithm solving the lower bound model (1)-(5). We consider the following two subproblems.

Subproblem (P1)

$$\frac{1}{m}\sum_{j\in\mathcal{J}} p_j \to \min,$$

$$\sum_{j\in\mathcal{J}} W_j^{\alpha} p_j^{1-\alpha} \le E,$$

$$p_j \ge 0,\ j\in\mathcal{J}.$$

Subproblem (P2)

$$\max_{j\in\mathcal{J}} \frac{p_j}{\delta_j} \to \min,$$

$$\sum_{j\in\mathcal{J}} W_j^{\alpha} p_j^{1-\alpha} \le E,$$

$$p_j \ge 0,\ j\in\mathcal{J}.$$

Problems (P1) and (P2) are relaxations of the problem (1)-(5). Therefore, if an optimal solution of (P1) or (P2) is a feasible solution of (1)-(5), it is also an optimal solution of (1)-(5).

Our algorithm for (1)-(5) consists of three steps. At the first two steps, problems (P1) and (P2) are solved, and at the third step, a combination of (P1) and (P2) is formed by equating the values of the objective functions.

The first step. Before solving problem (P1), we check the following condition

$$\max_{j\in\mathcal{J}} \frac{W_j}{\delta_j} \le \frac{1}{m}\sum_{j\in\mathcal{J}} W_j. \tag{16}$$

If inequality (16) takes place, then the optimal solution of problem (P1) is an optimal solution of problem (1)-(5), otherwise, go to the second step. Indeed, problem (P1) corresponds to the case when it is required to schedule jobs on one processor. So, in an optimal solution of (P1) all jobs have the same speed s, which can be found through equation

$$\sum_{j\in\mathcal{J}} W_j s^{\alpha-1} = E.$$

The processing times of jobs are

$$p_i = \frac{W_i}{s} = \frac{W_i\left(\sum_{j\in\mathcal{J}} W_j\right)^{\frac{1}{\alpha-1}}}{E^{\frac{1}{\alpha-1}}},\ i\in\mathcal{J}. \tag{17}$$

The condition (16) guarantees that the inequality $\max_{j\in\mathcal{J}} \frac{p_j}{\delta_j} \le \frac{1}{m}\sum_{j\in\mathcal{J}} p_j$ holds for job durations (17).

The second step. Initially we check the condition

$$\sum_{j\in\mathcal{J}} \delta_j \le m, \tag{18}$$

which indicates that the optimal solution of problem (P2) is an optimal solution of problem (1)-(5). Problem (P2) corresponds to simultaneous execution of the jobs. Thus, all jobs should have identical processing times on all utilized processors

$$\frac{p_i}{\delta_i} = \frac{\left(\sum_{j\in\mathcal{J}} \delta_j^{\alpha-1} W_j^{\alpha}\right)^{\frac{1}{\alpha-1}}}{E^{\frac{1}{\alpha-1}}},\ i \in \mathcal{J}. \tag{19}$$

The condition (18) guarantees that $\frac{1}{m}\sum_{j\in\mathcal{J}} p_j \le \max_{j\in\mathcal{J}} \frac{p_j}{\delta_j}$ for job durations (19). However, if the total number of processors required by all jobs is more then m, we go to the third step.

The third step. If neither inequality (16) nor inequality (18) is satisfied, according to the Karush-Kuhn-Tucker necessary and sufficient conditions, the following equality should hold for an optimal solution of problem (1)-(5):

$$\frac{1}{m}\sum_{j\in\mathcal{J}} p_j = \max_{j\in\mathcal{J}} \frac{p_j}{\delta_j}.$$

As a result we have problem (P3):

$$\sum_{j\in\mathcal{J}} p_j \to \min,$$
$$\max_{j\in\mathcal{J}} \frac{p_j}{\delta_j} - \frac{1}{m}\sum_{j\in\mathcal{J}} p_j = 0,$$
$$\sum_{j\in\mathcal{J}} W_j^{\alpha} p_j^{1-\alpha} \le E,$$
$$p_j \ge 0,\ j \in \mathcal{J}.$$

Solving (P3) begins with the optimal solution of (P1), where the processing times of jobs on one processor are equal to

$$p_i = \frac{W_i\left(\sum_{j\in\mathcal{J}} W_j\right)^{\frac{1}{\alpha-1}}}{E^{\frac{1}{\alpha-1}}},\ i \in \mathcal{J},$$

and the processing times of jobs on the maximum possible number of processors are $\frac{p_i}{\delta_i},\ i \in \mathcal{J}$.

As we can see, job durations $\frac{p_j}{\delta_j}$ are proportional to $\frac{W_j}{\delta_j}$. We order jobs in non-increasing $\frac{W_1}{\delta_1} \ge \frac{W_2}{\delta_2} \ge \dots \frac{W_n}{\delta_n}$. Let l denote the job with a minimal ratio $\frac{W_l}{\delta_l}$ such that

$$\frac{W_l}{\delta_l} > \frac{1}{m}\sum_{j\in\mathcal{J}} W_j. \tag{20}$$

This condition is equivalent to the condition $\frac{p_l}{\delta_l} > \frac{1}{m}\sum_{j\in\mathcal{J}} p_j$.

Because the objective of (P3) is a linear function, a necessary condition for the optimal solution of (P3) is

$$\frac{p_j}{\delta_j} = \frac{1}{m}\sum_{i\in\mathcal{J}} p_i,\ j = 1,\ldots,l.$$

Then we formulate a new problem (P4):

$$\sum_{j\in\mathcal{J}} p_j \to \min,$$

$$\frac{p_j}{\delta_j} = \frac{1}{m}\sum_{i\in\mathcal{J}} p_i,\ j = 1,\ldots,l,$$

$$\sum_{j\in\mathcal{J}} W_j^{\alpha} p_j^{1-\alpha} \le E,$$

$$p_j \ge 0,\ j \in \mathcal{J}.$$

In order to find an optimal solution of problem (P4) we use the Lagrangian method constructing the Lagrangian function

$$L(p_j,\lambda_j) = \sum_{j\in\mathcal{J}} p_j + \sum_{j=1}^{l}\lambda_j\left\{\frac{p_j}{\delta_j} - \sum_{i\in\mathcal{J}}\frac{p_i}{m}\right\} + \lambda_{l+1}\left(\sum_{j\in\mathcal{J}} W_j^{\alpha} p_j^{1-\alpha} - E\right).$$

We calculate the partial derivatives, equate them to zero and find the processing times of jobs

$$\frac{p_j}{\delta_j} = \frac{\sum_{i=l+1}^{n} W_i}{(m - \sum_{i=1}^{l}\delta_i)}C,\ j = 1,\ldots,l,$$

$$\frac{p_j}{\delta_j} = \frac{W_j}{\delta_j}C,\ j = l+1,\ldots,n,$$

where

$$C = \left(\frac{E}{\sum_{j=1}^{l}\frac{W_j^{\alpha}(\delta_j)^{1-\alpha}(\sum_{i=l+1}^{n} W_i)^{1-\alpha}}{(m-\sum_{i=1}^{l}\delta_i)^{1-\alpha}} + \sum_{j=l+1}^{n} W_j}\right)^{\frac{1}{1-\alpha}}.$$

Note that $m > \sum_{i=1}^{l}\delta_i$ due to condition (20) for choosing index l.

If the following inequality is satisfied for the obtained durations of jobs $l+1,\ldots,n$

$$\max_{j=l+1,\ldots,n}\frac{p_j}{\delta_j} = \frac{p_{l+1}}{\delta_{l+1}} \le \frac{1}{m}\sum_{j\in\mathcal{J}} p_j,$$

then we have an optimal solution of problem (1)-(5). Otherwise, we go to solving the problem (P4) with new value of l.

The time complexity of the method is $O\left(n\log\left(nm\right)\right)$ time if we have an oracle to exponentiation and root extraction operations.

References

[1] K. Belkhale, P. Banerjee, *An approximate algorithm for the partitionable independent task scheduling problem*, In: *Proceedings of International Conference on Parallel Processing (ICPP-90)*, **1** (1990), 72–75.

[2] M. Drozdowski, *Scheduling for parallel processing*, Springer-Verlag, London, 2009. Zbl 1187.68090

[3] M. Drozdowski, W. Kubiak, *Scheduling parallel tasks with sequential heads and tails*, Ann. Oper. Res., **90** (1999), 221–246. Zbl 0947.68017

[4] J. Du, J.Y-T. Leung, *Complexity of scheduling parallel task systems*, SIAM J. Discrete Math., **2**:4 (1989), 473–487. Zbl 0676.90029

[5] J. Glasgow, H. Shachnai, *Channel based scheduling of parallelizable tasks*, In *Proceedings Fifth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Los Alamitos, CA, USA, (1997), 11–16.

[6] K. Jansen, L. Porkolab. *Linear-time approximation schemes for scheduling malleable parallel tasks*, Algorithmica, **32**:3 (2002), 507–520. Zbl 1009.68013

[7] A. Kononov, Y. Kovalenko, *Approximation algorithms for energy-efficient scheduling of parallel jobs*, J. Sched., **23**:6 (2020), 693–709. Zbl 1456.90072

[8] A.V. Kononov, Yu.V. Zakharova, *Speed scaling scheduling of multiprocessor jobs with energy constraint and makespan criterion*, J. Glob. Optim., **83**:3 (2022), 539–564. Zbl 7550247

[9] R. Krishnamurti, B. Narahari, *An approximation algorithm for preemptive scheduling on parallel-task systems*, SIAM J. Discrete Math., **8**:4 (1995), 661–669. Zbl 0845.68009

[10] H. Kuhn, A. Tucker, *Nonlinear programming*, In: *Proc. Berkeley Sympos. math. Statist. Probability, California July 31 - August 12, 1950* (1951), 481–492. Zbl 0044.05903

[11] R. Lepère, D. Trystram, G. Woeginger, *Approximation algorithms for scheduling malleable tasks under precedence constraints*, Int. J. Found. Comput. Sci., **13**:4 (2002), 613–627. Zbl 1066.68010

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

[13] Y. Nesterov, *Lectures on convex optimization*, Springer, Cham, 2018. Zbl 1427.90003

[14] Q. Wang, K.H. Cheng, *List scheduling of parallel tasks*, Inf. Process. Lett., **37**:5 (1991), 291–297. Zbl 0724.68013

[15] Q. Wang, K.H. Cheng, *A heuristic of scheduling parallel tasks and its analysis*, SIAM J. Comput., **21**:2 (1992), 281–294. Zbl 0756.90053

Alexander Kononov
Sobolev Institute of Mathematics,
4, Koptyuga ave.,
Novosibirsk, 630090, Russia
Email address: `alvenko@math.nsc.ru`

Yulia Zakharova
Sobolev Institute of Mathematics,
4, Koptyuga ave.,
Novosibirsk, 630090, Russia,
Dostoevsky Omsk State University,
55a, Mira ave.,
Omsk, 644077, Russia
Email address: `kovalenko@ofim.oscsbras.ru`