Speed Scaling Scheduling of Multiprocessor Jobs with Energy Constraint and Total Completion Time Criterion

Alexander V. Kononov[1] **and Yulia V. Zakharova**[2]

[1]Sobolev Institute of Mathematics
Novosibirsk, Russia
alvenko@math.nsc.ru

[2]Sobolev Institute of Mathematics
Dostoevsky Omsk State University
Omsk, Russia
julia.kovalenko.ya@yandex.ru

ABSTRACT

We consider the problem of speed scaling scheduling multiprocessor jobs under the given energy budget. A multiprocessor job must be performed by more than one processor at a time. The speed of a processor may be different for different jobs. Running a job at a lower speed is more energy efficient. However, this takes a longer time and affects the performance. Our goal is to minimize the total completion time of jobs. We analyze the complexity of both parallel and dedicated statements of the problem. Approximation and exact algorithms are proposed for various cases. In our algorithms, a sequence of jobs and their processing time are calculated at the first stage, and then a feasible solution is constructed using a greedy rule of list type.

Keywords: multiprocessor job, speed scaling, scheduling, approximation algorithm, NP-hardness.

2000 Mathematics Subject Classification: 90B35, 68M20, 05A99.

1 Introduction

We consider the following speed scaling scheduling problem. Jobs from the set $\mathcal{J} = \{1, \dots, n\}$ must be executed on m speed scalable processors. Preemptions of jobs are disallowed. Let V_j denote the processing volume (work) V_j of job $j \in \mathcal{J}$. In the parallel statement of the problem, the number of the required processors $size_j$ are given for each job $j \in \mathcal{J}$. Such jobs are called rigid jobs and may be performed by any subset of processors of the indicated size (Drozdowski, 2009). In the dedicated version, we investigate single-mode multiprocessor jobs (Drozdowski, 2009), where a subset of processors fix_j is specified for a job $j \in \mathcal{J}$. The instances with moldable and malleable jobs (Drozdowski, 2009) are also analyzed. The job j is called moldable if it can be executed on arbitrary number of processors up to the given upper bound on the parallelization level. The number of used processors can be changed during the schedule for malleable jobs.

We suggest that the homogeneous model with continuous spectrum of processor speeds in speed-scaling takes place. Here if a processor works with a speed s, then instantaneous energy consumption (power) is s^α, where $\alpha > 1$ is a constant. The total energy consumption is power integrated over time. The speed of a processor may be different for different jobs. When m_j processors are utilized for execution of a job $j \in \mathcal{J}$, the total volume V_j is uniformly partitioned between processors, which run at the same speed. In this case $W_j := \frac{V_j}{m_j}$ denote the work of the job j on one processor.
Our goal is to construct a feasible schedule, that minimizes the sum of job completion times $\sum C_j$ and guarantees the summed energy consumption no more than a given budget E. This is a natural assumption when the battery energy is fixed, i.e., the scheduling problem has applications in computer devices whose lifespan depends on a limited battery efficiency (for example, multi-core laptops). Also, bi-criteria settings of minimizing a scheduling metric and energy consumption 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 may reasonably be estimated, so we bound the energy used, and optimize the regular timing criterion.
We use the following classic notations $P|size_j, energy|\sum C_j$, $P|any, \delta_j, energy|\sum C_j$, $P|var, \delta_j, energy|\sum C_j$ and $P|fix_j, energy|\sum C_j$ for the speed-scaling scheduling with the limited energy consumption of rigid, moldable, malleable and single-mode jobs, respectively.

2 Previous Research

Pruhs (Pruhs, Uthaisombut and Woeginger, 2008a) et al. studied the problem of minimization of the average flow time on a single processor under a fixed amount of energy and given release dates of jobs. For jobs with unit works, they proposed a polynomial-time algorithm that constructs a schedule with the minimum average execution time for each possible energy level. Bunde (Bunde, 2009) generalized this approach to the multi-processor case. $O(1)$-approximation algorithm allowing an additional factor of $(1+\varepsilon)$ energy was developed for scheduling jobs with arbitrary works on a single processor. Albers and Fujiwara (Albers and Fujiwara, 2007) considered online and offline versions of single-processor scheduling, where energy consumption plus job flow times is minimized. A deterministic constant competitive online algorithm and an offline dynamic programming algorithm were developed for unit-work jobs. In (Bansal, Pruhs and Stein, 2009), the authors give an online speed scaling algorithm that is O(1)-competitive for the objective of weighted flow time plus energy and arbitrary work jobs. In the first step, they relax the objective function to be fractional weighted flow plus energy and then solve the obtained problem. Then the solution obtained at the first step is transformed with the loss of a small factor in the competitive ratio.
Shabtay et al. (Shabtay and Kaspi, 2006) analyzed the closely related problem of scheduling single-processor jobs on identical parallel processors, where the durations of jobs p_j are resource-dependent in according with the following relation $p_j(R_j) = \left(\frac{W_j}{R_j}\right)^\kappa$. Here W_j is the workload of job j, R_j is the amount of resources allocated to process job j, $0 < \kappa \leq 1$ is a positive constant. An exact polynomial time algorithm was provided for multiprocessor non-preemptive cases with $\sum_j C_j$ criterion. The algorithm can be applied to speed scaling

scheduling of single-processor jobs.
Speed scaling scheduling with the makespan objective has been extensively researched. Various approaches to the construction of approximation algorithms for single-processor and multiprocessor jobs have been proposed (see, for example, (Kononov and Kovalenko, 2020; Bunde, 2009; Pruhs et al., 2008a)).
Now we present the known results for the parallel and multiprocessor jobs under given durations and without consideration of energy consumption. The problem for non-preemptive rigid jobs is strongly NP-hard even in the case of two processors (Lee and Cai, 1999), and it is NP-hard for preemptive settings when the number of processors is part of the input (Drozdowski and Dell'Olmo, 2000). Approximation algorithms with a constant factor approximation guarantee have been proposed for the non-preemptive cases. These algorithms use scheduling rules of list type (Turek, Ludwig, Wolf, Fleischer, Tiwari, Glasgow, Schwiegelshohn and Yu, 1994) and scheduling to minimize average response time (SMART) (Schwiegelshohn, Ludwig, Wolf, Turek and Yu, 1998). The problem of scheduling non-preemptive jobs of single-mode type on two processors is NP-hard (Hoogeveen, van de Velde and Veltman, 1994) and strongly NP-hard for two-processor jobs (Kubale, 1996). A reconstruction of a preemptive schedule to the non-preemptive one yields a 2-approximation algorithm for two-processor instances (Cai, Lee and Li, 1998), and First Fit Coloring approach provides a 2-approximate solution for two-processor jobs with unit works (Giaro, Kubale, Maiafiejski and Piwakowski, 1999).
Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient for solutions of convex optimization problems to be optimal (Kuhn and Tucker, 1951; Boyd and Vandenberghe, 2004). The Karush-Kuhn-Tucker conditions are often applied in algorithmic technics for scheduling problems and convex resource consumption functions. The classic Karush-Kuhn-Tucker conditions allow the identification of the structural properties of optimal schedules and developing combinatorial algorithms. See examples in (Bampis, Letsios and Lucarelli, 2014; Shabtay and Kaspi, 2006; Angel, Bampis, Kacem and Letsios, 2012; Pruhs, Uthaisombut and Woeginger, 2008b; Bansal and Pruhs, 2005; Bampis, Letsios, Milis and Zois, 2016) for problems with convex resource consumption functions and various criteria. We use Karush-Kuhn-Tucker conditions for solving auxiliary convex programs to compute bounds on the criteria in NP-hardness proofs and approximation algorithms.

Our results.

We prove NP-hardness of scheduling problems $P|size_j \le \frac{m}{2}, energy|\sum C_j$ and $P|fix_j,\ |fix_j| = 2, energy|\sum C_j$, and develop two-stage approximation or exact algorithms for the following particular cases:

- rigid jobs,
- moldable jobs,
- two-processors parallel and dedicated instances,
- single-processor instances,
- fully-parallelizable jobs,

- incompatible jobs.

At the first stage, we compute a lower bound on the total execution time and find a sequence and durations of jobs using a convex program of specific structure. Then, at the second stage, we reformulate our problem as a classical scheduling problem without speed scaling and apply ”list-scheduling“ methods and more complex approaches to construct feasible solutions. Whenever a subset of processors is idle, the ”list-scheduling“ method schedules the first job in the list that does not require more processors than are available.

3 Rigid Jobs

In this section, we consider rigid jobs. Firstly, we prove that the problem is NP-hard. Secondly, we present an approximation algorithm for the instances, where sizes and workloads of jobs are agreeable.

3.1 NP-hardness

Theorem 3.1. *Problem $P|size_j \leq \frac{m}{2}, W_j = 1, energy|\sum C_j$ is NP-hard in the strong sense.*

Proof. We prove that the strongly NP-complete 3-PARTITION problem can be reduced to the decision version of problem $P|size_j \leq \frac{m}{2}, W_j = 1, energy|\sum C_j$.
Let we are given an instance of the 3-PARTITION problem: a set of $3q$ elements with weights $a_j,\ j = 1, \dots, 3q$, where $\sum_{j=1}^{3q} a_j = Bq$ and $\frac{B}{4} < a_j < \frac{B}{2}$. Could the set be partitioned into q subsets $A_1, \dots, A_q$ such that $\sum_{j \in A_i} a_j = B$?
An instance of $P|size_j, energy|\sum C_j$ is constructed as follows. Put the number of jobs $n = 3q$, the number of processors $m = B$, and the energy budget $E = Bq$. For every a_j we generate a job $j,\ j = 1, \dots, 3q$. We set $W_j = 1$, $size_j = a_j$, $V_j = a_j$ for $j \in \mathcal{J}$. In the decision version of $P|size_j, W_j = 1, energy|\sum C_j$ it is required to answer the question: Is there a schedule with $\sum C_j$ value not greater than a given threshold T?
To determine the value of T we solve an auxiliary problem with $\sum_{j=1}^{3q} a_j$ single-processor jobs of the unit works, i.e. each rigid job is replaced by $size_j$ single-processor jobs. Such problem has the unique optimal solution (with the accuracy of placing jobs on processors and permuting jobs on each processor), where each processor executes q jobs and uses energy budget q. Now we find optimal durations of jobs on each processor, using the following convex model:

$$\sum_{j=1}^{q} p_j(q - j + 1) \to \min, \tag{3.1}$$

$$\sum_{j=1}^{q} p_j^{1-\alpha} = q, \tag{3.2}$$

$$p_j \geq 0,\ j = 1, \dots, q. \tag{3.3}$$

Here p_j is the duration of j-th job on a processor, $j = 1, \dots, q$.
We compose the Lagrangian function $L(p_j, \lambda) = \sum_{j=1}^{q} p_j(q - j + 1) + \lambda \left(\sum_{j=1}^{q} p_j^{1-\alpha} - q \right)$. Equate partial derivatives to zero:

$$\frac{\partial L}{\partial p_j} = (q-j+1) + \lambda(1-\alpha)p_j^{-\alpha} = 0,\ p_j = \left(\frac{\lambda(\alpha-1)}{q-j+1}\right)^{1/\alpha},\ j = 1,\dots,q.$$

Place this expression into (3.2) and obtain

$$\sum_{j=1}^{q}\left(\frac{\lambda(\alpha-1)}{q-j+1}\right)^{1-\alpha/\alpha} = q,\ (\lambda(\alpha-1))^{1/\alpha} = \left(\frac{\sum_{j=1}^{q}(q-j+1)^{\alpha-1/\alpha}}{q}\right)^{1/\alpha-1}.$$

Then we calculate the optimal solution:

$$p_j^* = \frac{\left(\sum_{j=1}^{q}(q-j+1)^{\frac{\alpha-1}{\alpha}}\right)^{\frac{1}{\alpha-1}}}{q^{\frac{1}{\alpha-1}}(q-j+1)^{\frac{1}{\alpha}}},\ j = 1,\dots,q,$$

$$\sum C_j^* = \sum_{j=1}^{q} p_j(q-j+1) = \left(\sum_{j=1}^{q}(n-j+1)^{\frac{\alpha-1}{\alpha}}\right)^{\frac{\alpha}{\alpha-1}} q^{\frac{1}{1-\alpha}}.$$

Note that each next job has more duration than the previous one. The optimal schedule for each processor does not have idle times. The optimal total completion time for all processors is equal to $m\sum C_j^*$. Set the threshold $T := 3\sum C_j^*$, since at most three rigid jobs can be performed simultaneously.

We show that a positive answer to the constructed problem $P|size_j, energy|\sum C_j$ with $\sum C_j \leq T$ implies a positive answer to 3-PARTITION and otherwise. Indeed, $\sum C_j \geq 3\sum C_j^*$ for any feasible schedule of the stated instance due to the previously presented relaxation to single-processor jobs with the same total work. So, a feasible schedule with $\sum C_j = T$ exists if and only if all processors perform q jobs, and every three jobs, executed in parallel and using all available processors, correspond to three elements forming set A_i such that $\sum_{j\in A_i} a_j = B$, $i = 1,\dots,q$. This schedule is similar to the optimal one of the corresponding problem with single-processor jobs (see example in Fig. 1).

The presented reduction is polynomial. So, problem $P|size_j \leq \frac{m}{2}, W_j = 1, energy|\sum C_j$ is strongly NP-hard. $\square$

Now we go to construct an approximation algorithm for the considered problem with rigid jobs satisfying specific property on the volumes and sizes. A problem-dependent convex program allows us to calculate speeds of jobs. We will use the "list-scheduling" method to find a feasible solution with approximation guarantee.

3.2 Approximation Algorithm

The order of jobs, and as a result the completion times of all jobs play important role in the problems with the total completion time objective. So, initially we consider the case, when the order of job starting times is fixed, and calculate a lower bound on the scheduling metric ignoring rectangle nature of the jobs. Assume that the jobs are placed in a schedule according to the permutation $\pi = (\pi_1,\dots,\pi_n)$. Using the lower bound on the total completion time provided in (Turek et al., 1994) for rigid jobs with the given durations, we formulate the following convex program:

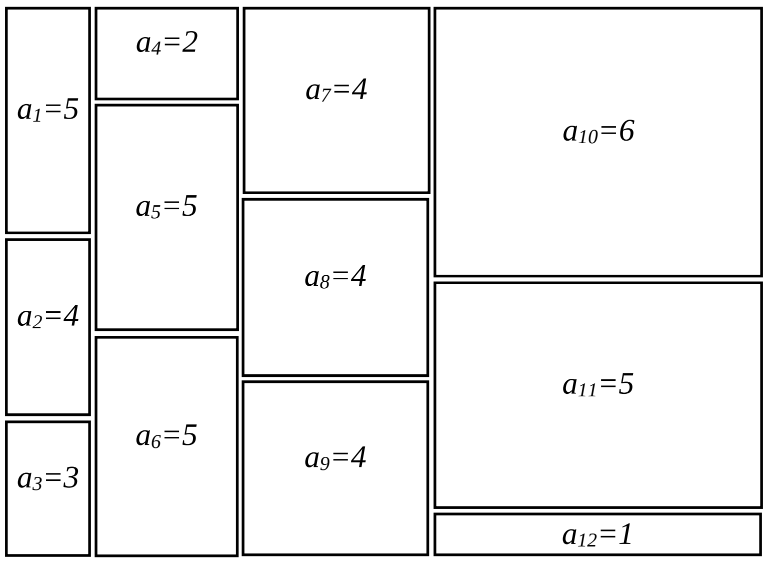

Figure 1: A feasible schedule corresponding to the positive answer of 3-Partition problem: $A_1 = \{1, 2, 3\}$, $A_2 = \{4, 5, 6\}$, $A_3 = \{7, 8, 9\}$, $A_4 = \{10, 11, 12\}$, $\sum_{j\in A_i} a_j = 12$.

$$LB(\pi) = \frac{1}{m}\sum_{j=1}^{n}\sum_{i=1}^{j} size_{\pi_i} p_{\pi_i} + \frac{1}{2}\sum_{j=1}^{n} p_{\pi_j} - \frac{1}{2m}\sum_{j=1}^{n} size_{\pi_j} p_{\pi_j} \to \min, \tag{3.4}$$

$$\sum_{j=1}^{n} W_j^{\alpha} p_j^{1-\alpha} size_j \leq E. \tag{3.5}$$

$$p_j \geq 0,\ j \in \mathcal{J}. \tag{3.6}$$

Here π_i is the i-th job in accordance with permutation π, p_{π_i} is the execution time of job π_i on each of the utilized processors. Recall that $W_i = \frac{V_i}{size_i}$ for rigid jobs.

Lemma 3.2. *The program (3.4)-(3.6) for permutation π of jobs is polynomially solvable and the optimal objective*

$$LB^*(\pi) = \frac{E^{\frac{1}{1-\alpha}}}{m}\left(\sum_{i=1}^{n} W_{\pi_i} size_{\pi_i}\left(n - i + 0.5 + \frac{0.5m}{size_{\pi_i}}\right)^{\frac{\alpha-1}{\alpha}}\right)^{\frac{\alpha}{\alpha-1}}. \tag{3.7}$$

Proof. We solve program (3.4)-(3.6) by means of the Lagrangian method. Define the Lagrangian function $L(p_{\pi_j}, \lambda)$ as

$$L(p_{\pi_j}, \lambda) = \frac{1}{m}\sum_{j=1}^{n}\sum_{i=1}^{j} size_{\pi_i} p_{\pi_i} + \frac{1}{2}\sum_{j=1}^{n}\left(1 - \frac{size_{\pi_j}}{m}\right) p_{\pi_j} +$$
$$\lambda\left(\sum_{j=1}^{n} W_{\pi_j}^{\alpha} p_{\pi_j}^{1-\alpha} size_{\pi_j} - E\right).$$

Because the objective (3.4) is linear function and the constraint (3.5) is represented by convex function, the necessary and sufficient conditions for an optimal solution are (partial derivatives are equal to zero) (Kuhn and Tucker, 1951; Boyd and Vandenberghe, 2004):

$$\frac{\partial L}{\partial p_{\pi_i}} = \frac{1}{m} size_{\pi_i}(n-i+1) + \frac{1}{2}\left(1 - \frac{size_{\pi_i}}{m}\right) + \lambda W_{\pi_i}^{\alpha} size_{\pi_i}(1-\alpha) p_{\pi_i}^{-\alpha} = 0,$$
$$i = 1, \ldots, n.$$

Rewriting the expressions, we obtain

$$p_{\pi_i} = ((\alpha - 1)\lambda m)^{\frac{1}{\alpha}} \frac{W_{\pi_i}}{\left(n - i + 0.5 + \frac{0.5m}{size_{\pi_i}}\right)^{\frac{1}{\alpha}}}, \tag{3.8}$$
$$i = 1, \ldots, n.$$

The processing times (3.8) are placed into equation

$$\frac{\partial L}{\partial \lambda} = \sum_{j=1}^{n} W_j^{\alpha} p_j^{1-\alpha} size_j - E = 0.$$

This gives us the following equation

$$((\alpha - 1)\lambda m)^{\frac{1-\alpha}{\alpha}} \sum_{j=1}^{n} W_{\pi_j} size_{\pi_j} \left(n - j + 0.5 + \frac{0.5m}{size_{\pi_j}}\right)^{\frac{\alpha-1}{\alpha}} = E.$$

As a result, we calculate the value

$$((\alpha - 1)\lambda m)^{\frac{1}{\alpha}} = E^{\frac{1}{1-\alpha}} \left(\sum_{j=1}^{n} W_{\pi_j} size_{\pi_j} \left(n - j + 0.5 + \frac{0.5m}{size_{\pi_j}}\right)^{\frac{\alpha-1}{\alpha}}\right)^{\frac{1}{\alpha-1}}$$

and the durations of jobs from (3.8)

$$p_{\pi_i} = \frac{E^{\frac{1}{1-\alpha}} W_{\pi_i}}{\left(n - i + 0.5 + \frac{0.5m}{size_{\pi_i}}\right)^{\frac{1}{\alpha}}} \cdot \left(\sum_{j=1}^{n} \frac{W_{\pi_j} size_{\pi_j}}{\left(n - j + 0.5 + \frac{0.5m}{size_{\pi_j}}\right)^{\frac{1-\alpha}{\alpha}}}\right)^{\frac{1}{\alpha-1}}, \tag{3.9}$$
$$i = 1, \ldots, n.$$

The obtained values for execution times are placed in expression (3.4) and the lower bound on $\sum C_j$ for an arbitrary permutation π of jobs is calculated as follows

$$LB^*(\pi) = \frac{E^{\frac{1}{1-\alpha}}}{m} \left(\sum_{i=1}^{n} W_{\pi_i} size_{\pi_i} \left(n - i + 0.5 + \frac{0.5m}{size_{\pi_i}}\right)^{\frac{\alpha-1}{\alpha}}\right)^{\frac{\alpha}{\alpha-1}}.$$

$\square$

Lemma 3.3. *Let the non-decreasing order of total works V_j corresponds to the non-decreasing order of $size_j$ (i.e. $V_i < V_j$ implies $size_i \le size_j$ for $i \neq j \in \mathcal{J}$). Then the minimum value of $LB^*(\pi)$ is reached on the permutation, where the jobs are ordered by non-decreasing of the required processors numbers $size_j$.*

Proof. The minimization of (3.7) is equivalent to the minimization of

$$G(\pi) = \sum_{i=1}^{n} V_{\pi_i} \left(n - i + 0.5 + \frac{0.5m}{size_{\pi_i}} \right)^{\frac{\alpha-1}{\alpha}}.$$

over all possible permutations π. Let π^* be the the permutation of jobs in non-decreasing of $size_i$-values. Consider two consecutive jobs in π^*, say π^*_i and π^*_{i+1}, for which $size_{\pi^*_i} = x$, $size_{\pi^*_{i+1}} = y$, $V_{\pi^*_i} = V_x$, $V_{\pi^*_{i+1}} = V_y$ and $x \leq y$, $V_x \leq V_y$. These jobs give

$$g(\pi^*, i) = V_x \left(n - i + 0.5 + \frac{0.5m}{x} \right)^{\frac{\alpha-1}{\alpha}} + V_y \left(n - i - 0.5 + \frac{0.5m}{y} \right)^{\frac{\alpha-1}{\alpha}}$$

in $G(\pi^*)$. Suppose $\pi' = (\pi^*_1, \ldots, \pi^*_{i-1}, \pi^*_{i+1}, \pi^*_i, \pi^*_{i+2}, \ldots, \pi^*_n)$ and

$$g(\pi', i) = V_y \left(n - i + 0.5 + \frac{0.5m}{y} \right)^{\frac{\alpha-1}{\alpha}} + V_x \left(n - i - 0.5 + \frac{0.5m}{x} \right)^{\frac{\alpha-1}{\alpha}}.$$

Then

$$g(\pi^*, i) - g(\pi', i) = V_x \left(\left(n - i + 0.5 + \frac{0.5m}{x} \right)^{\frac{\alpha-1}{\alpha}} - \left(n - i - 0.5 + \frac{0.5m}{x} \right)^{\frac{\alpha-1}{\alpha}} \right)$$

$$-V_y \left(\left(n - i + 0.5 + \frac{0.5m}{y} \right)^{\frac{\alpha-1}{\alpha}} - \left(n - i - 0.5 + \frac{0.5m}{y} \right)^{\frac{\alpha-1}{\alpha}} \right) \leq$$

$$(V_x - V_y) \left(\left(n - i + 0.5 + \frac{0.5m}{y} \right)^{\frac{\alpha-1}{\alpha}} - \left(n - i - 0.5 + \frac{0.5m}{y} \right)^{\frac{\alpha-1}{\alpha}} \right) \leq 0.$$

The first inequality follows from Lemma 3.4. Therefore, $G(\pi^*) \leq G(\pi')$. $\square$

Lemma 3.4. *Function* $f(x) = \left(a + \frac{b}{x}\right)^{\frac{\alpha-1}{\alpha}} - \left(a - 1 + \frac{b}{x}\right)^{\frac{\alpha-1}{\alpha}}$ *is increasing for* $x \geq 1$, $\alpha > 1$, $a > 1$ *and* $b > 0$.

Proof. Compute derivative $f'(x) = -\frac{b}{x^2} \cdot \frac{\alpha-1}{\alpha} \left(a + \frac{b}{x}\right)^{\frac{-1}{\alpha}} + \frac{b}{x^2} \cdot \frac{\alpha-1}{\alpha} \left(a - 1 + \frac{b}{x}\right)^{\frac{-1}{\alpha}} = \frac{b}{x^2} \cdot \frac{\alpha-1}{\alpha} \left(\left(a - 1 + \frac{b}{x}\right)^{\frac{-1}{\alpha}} - \left(a + \frac{b}{x}\right)^{\frac{-1}{\alpha}} \right)$. It is easy to see that $f'(x) > 0$. $\square$

From now on, we suppose that the non-decreasing order of V_j corresponds to the non-decreasing order of $size_j$ (i.e. workloads and sizes of jobs are agreeable). Let the jobs are ordered by non-decreasing of $size_j$-values. We denote by $\bar{p}_j$ the durations of jobs, and by $LB(\bar{p}_j) = \sum_{j=1}^{n} \left(\frac{1}{m} \sum_{i=1}^{j-1} size_i \bar{p}_i + \frac{1}{2} \bar{p}_j + \frac{1}{2m} size_j \bar{p}_j \right)$ the lower bound corresponding to the optimal solution of problem (3.4)-(3.6).

We construct a schedule using the "list-scheduling" algorithm: The first job is scheduled at time 0. The next job is scheduled at the earliest time such that there are enough processors to execute it (see example in Fig. 2). The running time of the algorithm is $O(n^2)$.

Lemma 3.5. *Let* $V_i < V_j$ *implies* $size_i \leq size_j$ *for* $i \neq j \in \mathcal{J}$. *The "list-scheduling" algorithm generates a feasible schedule* S *with the total completion time at most* $2LB(\bar{p}_j)$ *for problem* $P|size_j \leq \frac{m}{2}, energy|\sum C_j$.

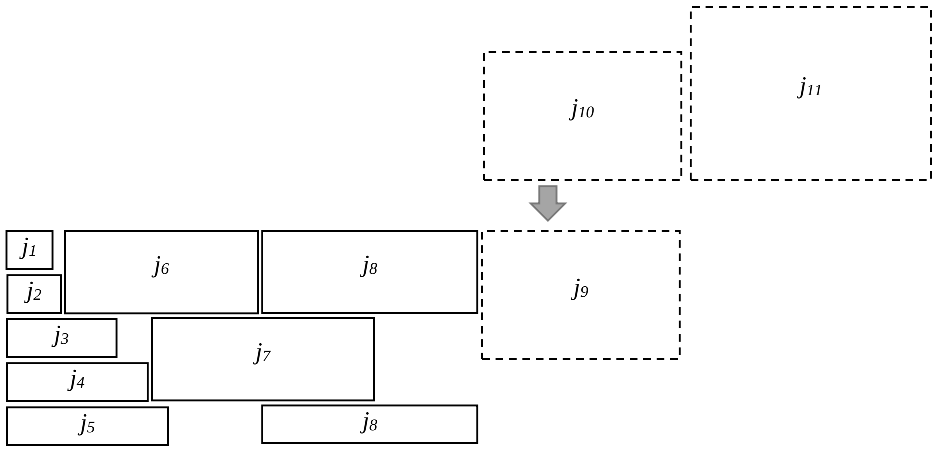

Figure 2: A part of the schedule constructed by the “list-scheduling” algorithm.

Proof. Each job utilizes at most half of processors. So, the starting time of job j is not greater than $2\left(\frac{1}{m}\sum_{i=1}^{j-1} size_i\bar{p}_i\right)$, as at least $\frac{m}{2}$ processors are busy at each time moment in schedule S. The completion time of job j satisfies condition $C_j \le \frac{2}{m}\sum_{i=1}^{j-1} size_i\bar{p}_i + \bar{p}_j$. In the sum we have

$$\sum_{j\in\mathcal{J}} C_j \le 2\left(\frac{1}{m}\sum_{j=1}^{n}\sum_{i=1}^{j-1} size_i\bar{p}_i + \frac{1}{2}\sum_{j=1}^{n}\bar{p}_j\right).$$

We compare this value with the lower bound $LB(\bar{p}_j)$ and conclude that $\sum_{j\in\mathcal{J}} C_j \le 2LB(\bar{p}_j)$. □

Lemma 3.6. *Let $W_i < W_j$ implies $size_i \le size_j$ for $i \ne j \in \mathcal{J}$. The “list-scheduling” algorithm generates a feasible schedule S with the total completion time at most $2LB(\bar{p}_j)$ for problem $P|size_j, energy|\sum C_j$.*

Proof. Note that the durations of jobs $\bar{p}_j$ do not decrease in the stated case (see formula (3.9)). If at least $\frac{m}{2}$ processors are busy at any time instance in S, then we use the same properties as in the previous Lemma.
Otherwise, assume that l is the last job, that requires less than $\frac{m}{2}$ processors. By the construction of S, less than $\frac{m}{2}$ processors may be busy only in interval $I := [C_l - \bar{p}_l, C_{l+1} - \bar{p}_{l+1})$. We denote the length of interval I by Δ, and let h be the minimal number of processors utilized in I. Note that $size_l \le h < \frac{m}{2}$ because job l is performed during all interval I.
It is easy to see that at every point in time during interval $[0, C_l - p_l)$ schedule S uses at least $m - size_l + 1 \ge m - h + 1$ processors. (Otherwise, job l should be started earlier.) Moreover, at least $m - h + 1$ processors are utilized in interval $[C_{l+1} - \bar{p}_{l+1}, C_n)$ as job $(l+1)$ is not executed in interval I and $size_j \ge size_{l+1}$ for all $j > l+1$.
Consider a job $j > l$. The completion time of this job is not greater than

$$\frac{1}{m-h+1}\left(\sum_{i=1}^{j-1} size_i\bar{p}_i - \Delta h\right) + \Delta + \bar{p}_j.$$

Now we prove that $\Delta - \frac{h}{m-h+1}\Delta + \bar{p}_j \le 2\left(\frac{1}{2}\bar{p}_j + \frac{size_j\bar{p}_j}{2m}\right)$. Indeed, due to $\Delta \le \bar{p}_l < \bar{p}_j$, $size_j + h > m$ and $m > 2h$ we have

$$\Delta - \frac{h}{m-h+1}\Delta + \bar{p}_j < \left(\frac{m-2h+1}{m-h+1}+1\right)\bar{p}_j = \frac{2m-3h+2}{m-h+1}\bar{p}_j =$$
$$\frac{(2m-3h+2)(2m)}{(m-h+1)(2m-h)} \cdot \frac{(2m-h)}{(2m)}\bar{p}_j = \frac{4m^2-6mh+4m}{2m^2-3mh+2m+(h^2-h)}\left(\frac{1}{2}+\frac{m-h}{2m}\right)\bar{p}_j \leq$$
$$2\left(\frac{1}{2}+\frac{m-h}{2m}\right)\bar{p}_j < 2\left(\frac{1}{2}\bar{p}_j + \frac{size_j\bar{p}_j}{2m}\right).$$

Therefore, the completion time of job $j > l$ is less then $2\left(\frac{1}{m}\sum_{i=1}^{j-1} size_i\bar{p}_i + \frac{1}{2}\bar{p}_j + \frac{size_j\bar{p}_j}{2m}\right)$. Moreover, the completion time of job $j \leq l$ is not greater than $\frac{2}{m}\sum_{i=1}^{j-1} size_i\bar{p}_i + \bar{p}_j$ as at least $m - size_l + 1 > \frac{m}{2}$ are busy in interval $[0, C_l - p_l)$. In the sum of completion times over all jobs we have
$$\sum_{j\in\mathcal{J}} C_j < 2\sum_{j=1}^{n}\left(\frac{1}{m}\sum_{i=1}^{j-1} size_i\bar{p}_i + \frac{1}{2}\bar{p}_j + \frac{size_j\bar{p}_j}{2m}\right).$$

□

Therefore, the following theorems take place.

Theorem 3.7. *A 2-approximate schedule can be found in polynomial time for problem $P|size_j \leq \frac{m}{2}, energy|\sum C_j$, in which the non-decreasing order of V_j corresponds to the non-decreasing order of $size_j$.*

Theorem 3.8. *A 2-approximate schedule can be found in polynomial time for scheduling problem $P|size_j, energy|\sum C_j$, in which the non-decreasing order of $W_j = \frac{V_j}{size_j}$ corresponds to the non-decreasing order of $size_j$.*

When $size_i \leq size_j$ implies $W_i < W_j$, then we also have $V_i = W_i size_i < V_j = size_j W_j$, but the converse statement is not true in the general case. So, the condition on the input parameters in Theorem 3.8 guarantees the condition in Theorem 3.7, but not vice versa.

3.3 Two-processor Instances

Here we consider the two-processor problem $P2|size_j, energy|\sum C_j$ with arbitrary workloads of jobs and propose a 2-approximation algorithm. Our algorithm has the following main idea. Given an instance of $P2|size_j, energy|\sum C_j$, we construct a single processor problem $1|energy|\sum C_j$, such that the optimal objective value is a lower bound for that of problem $P2|size_j, energy|\sum C_j$. Then we construct a feasible solution for the initial two-processor problem, using the same order of job starting times as that in the optimal sequence for $1|energy|\sum C_j$, but modifying processing times of jobs.

Let $(P2)$ denote the original problem $P2|size_j, energy|\sum C_j$ with job-volumes W_j and energy budget E. Formulate a single-processor problem $(P1)$ with job-volumes W'_j and energy budget E' constructed from $(P2)$ by letting $W'_j = \frac{W_j}{2}$ for single-processor jobs, $W'_j = W_j$ for two-processor jobs and $E' = \frac{E}{2}$. Since $(P1)$ is a single-processor problem, the ordering of jobs in non-decreasing volumes corresponds to an optimal solution, which can be found in polynomial time (see Subsection 6.1).

The following lemma shows that the optimal objective value $\sum C^*_j(P1)$ of $(P1)$ is a lower bound on the optimal objective value $\sum C^*_j(P2)$ of $(P2)$.

Lemma 3.9. $\sum C_j^*(P1) \leq \sum C_j^*(P2)$.

Proof. Let $S2$ be an arbitrary solution of $(P2)$. Construct a solution $S1$ of $(P1)$, decreasing the durations of single-processor jobs in two times and non-changing processing times of two-processor jobs. So the energy consumption will be not greater than $E/2 = E'$, and all job volumes will be processed. Moreover, $S1$ is generated such that the completion time $C_j(S1)$ of any job j in $S1$ is not greater than that $C_j(S2)$ in $S2$. Therefore, $\sum C_j^*(P1) \leq \sum_{j\in\mathcal{J}} C_j(S1) \leq \sum_{j\in\mathcal{J}} C_j(S2)$ for any schedule $S2$, in particular for the optimal solution of $(P2)$. Hence $\sum C_j^*(P1) \leq \sum C_j^*(P2)$. □

Approximation Algorithm $\mathcal{A}$.

Step 1: Given an instance $(P2)$, we generate the instance $(P1)$ as described above, reindex jobs in non-decreasing of volumes W_j', and find optimal durations p_j'.

Step 2: Calculate processing times of jobs for $(P2)$: $p_j = 2p_j'$ for single-processor jobs and $p_j = p_j'$ for two-processor jobs. Assign job j to the first available processor if it uses one processor, or to two processors when both of them are available if it requires two processors, keeping the same jobs order as obtained in Step 1.

Lemma 3.10. $\sum_{j\in\mathcal{J}} C_j(\mathcal{A}) \leq 2\sum C_j^*(P2)$.

Proof. Let $C_j(P1)$ be the completion time of job j for $(P1)$ in Step 1, and $C_j(P2)$ be the completion time of j for $(P2)$ in Step 2. By Lemma 3.9, we have $\sum_{j=1}^n C_j(P1) = \sum C_j^*(P1) \leq \sum C_j^*(P2)$. Since in Step 2, we keep the same jobs order as obtained in Step 1, $C_j(P2) \leq \sum_{i=1}^j p_i$ for each job j. Note that $C_j(P1) = \sum_{i=1}^j p_i' \geq \sum_{i=1}^j \frac{p_i}{2} \geq \frac{1}{2}C_j(P2)$. Thus $\sum_{j=1}^n C_j(P2) \leq 2\sum_{j=1}^n C_j(P1) \leq 2\sum C_j^*(P2)$. □

Theorem 3.11. *A 2-approximate schedule can be found in $O(n\log n)$ time for scheduling problem $P2|size_j, energy|\sum C_j$.*

4 Moldable Jobs

In this section, we consider the scheduling instances with moldable jobs. Recall that V_j denote the total processing volume of job j, i.e. the execution time of this job on one processor with unit speed. Let $\delta_j \leq m$ be the maximal possible number of processors, that may be utilized by job j. We consider the linear case, when the runtime of a job decreases linearly with the number of processors assigned to it.

Suppose that the non-decreasing order of total works V_j corresponds to the non-decreasing order of δ_j (i.e. $V_i < V_j$ implies $\delta_i \leq \delta_j$). In order to obtain a lower bound on the sum of completion times, we formulate the following convex model in the case of the given sequence π of jobs

$$\frac{1}{m}\sum_{j=1}^n\sum_{i=1}^j p_{\pi_i} + \frac{1}{2}\sum_{j=1}^n \frac{p_{\pi_j}}{\delta_{\pi_j}} - \frac{1}{2m}\sum_{j=1}^n p_{\pi_j} \to \min, \tag{4.1}$$

$$\sum_{j=1}^n V_j^\alpha p_j^{1-\alpha} \leq E, \tag{4.2}$$

$$p_j \geq 0,\ j \in \mathcal{J}. \tag{4.3}$$

Here p_j is the duration of job j on one processor in the total volume V_j. The duration of job j on $m_j \leq \delta_j$ processors will be equal to $\frac{p_j}{m_j}$.
Using the same properties as in Lemma 3.3 from Subsection 3.2, we can show that the minimum of (4.1), say

$$LB(\pi) := \frac{E^{1/1-\alpha}}{m}\left(\sum_{j=1}^{n} V_{\pi_j}\left(n-j+0.5+\frac{0.5m}{\delta_{\pi_j}}\right)^{\alpha-1/\alpha}\right)^{\alpha/\alpha-1},$$

is reached on the permutation π^*, where the jobs are ordered by non-decreasing of the maximum processors numbers $\delta_j,\ j \in \mathcal{J}$, The corresponding duration of a job on one processor in the total volume will be equal to

$$p^*_{\pi^*_i} = \frac{E^{\frac{1}{1-\alpha}} V_{\pi^*_i}}{\left(n-i+0.5+\frac{0.5m}{\delta_{\pi^*_i}}\right)^{\frac{1}{\alpha}}} \cdot \left(\sum_{j=1}^{n} \frac{V^*_{\pi_j}}{\left(n-j+0.5+\frac{0.5m}{\delta_{\pi^*_j}}\right)^{\frac{1-\alpha}{\alpha}}}\right)^{\frac{1}{\alpha-1}}, \qquad (4.4)$$
$$i = 1, \ldots, n.$$

Assume that the non-decreasing order of $\frac{V_j}{\delta_j}$ corresponds to the non-decreasing order of δ_j or the non-decreasing order of V_j corresponds to the non-decreasing order of δ_j and all $\delta_j \leq \frac{m}{2}$. 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 "list-scheduling" algorithm with processing times $\frac{p^*_j}{\delta_j},\ j \in \mathcal{J}$ and sequence of jobs π^*, then we obtain a 2-approximate solution as in the case of rigid jobs.
Now we show that a 2-approximate solution can be found for the more general case, when the non-decreasing order of V_j corresponds to the non-decreasing order of δ_j and δ_j are arbitrary. We assign the number of processors m_j for jobs as follows

$$m_j = \begin{cases} \delta_j & \text{if } \delta_j < \lceil \frac{m}{2} \rceil, \\ \lceil \frac{m}{2} \rceil & \text{if } \delta_j \geq \lceil \frac{m}{2} \rceil, \end{cases}$$

and construct a schedule using the "list-scheduling" algorithm based on the order of jobs in non-decreasing of $\delta_j,\ j \in \mathcal{J}$. Let us prove that the total completion time $\sum C_j(p^*_j) \leq 2LB(\pi^*)$. Indeed, at least $\frac{m}{2}$ processors are busy at each time moment in the schedule, so

$$\sum C_j(p^*_j) \leq \frac{2}{m}\sum_{j=1}^{n}\sum_{i=1}^{j-1} p^*_{\pi^*_i} + \sum_{j=1}^{n}\frac{p^*_{\pi^*_j}}{m_j} = \frac{2}{m}\sum_{j=1}^{n}\sum_{i=1}^{j} p^*_{\pi^*_i} + \sum_{j=1}^{n}\frac{p^*_{\pi^*_j}}{m_j} - \frac{2}{m}\sum_{j=1}^{n} p^*_{\pi^*_j} =$$
$$\frac{2}{m}\sum_{j=1}^{n}\sum_{i=1}^{j} p^*_{\pi^*_i} + \sum_{j=1}^{n} p^*_{\pi^*_j}\left(\frac{1}{m_j}-\frac{2}{m}\right) \leq \frac{2}{m}\sum_{j=1}^{n}\sum_{i=1}^{j} p^*_{\pi^*_i} + \sum_{j=1}^{n} p^*_{\pi^*_j}\left(\frac{1}{\delta_j}-\frac{1}{m}\right) \leq 2LB(\pi^*).$$

Therefore, we obtain

Theorem 4.1. *A 2-approximate schedule can be found in polynomial time for scheduling problem $P|any, \delta_j, energy|\sum C_j$, in which the non-decreasing order of V_j corresponds to the non-decreasing order of δ_j.*

We note here, that the complexity status of speed scaling scheduling problem $P|any, V_j = V, \delta_j, energy|\sum C_j$ is open.

5 Single Mode Multiprocessor Jobs

In this section, we consider single-mode multiprocessor jobs. Firstly, we prove that the problem is NP-hard. Secondly, we provide a constant factor approximation algorithm for two-processor instances.

5.1 NP-hardness

Theorem 5.1. *Problem $P|fix_j,\ |fix_j| = 2, V_j = 2, energy|\sum C_j$ is NP-hard in the strong sense.*

Proof. The proof is similar to the proof of Theorem 3.1, but it is based on the polynomial reduction of the strongly NP-complete Chromatic Index problem for cubic graphs (Holyer, 1981). The chromatic index of a graph is the minimum number of colors required to color the edges of the graph (form color classes) in such a way that no two adjacent edges have the same color. Consider an instance of the Chromatic Index problem on a cubic graph $G = (V, A)$, which asks whether the chromatic index $\chi'(G)$ is three. It is well-know that $\chi'(G) = 3$ or 4, and $|V|$ is even. Moreover, $\chi'(G) = 3$ if and only if each color class has exactly $\frac{1}{2}|V|$ edges.
We form an instance of $P|fix_j, energy|\sum C_j$ as follows. Put the number of jobs $n = |A|$, the number of processors $m = |V|$ and the energy budget $E = 2|A| = 3|V|$. Vertices correspond to processors. For every edge $\{u_j, v_j\}$ we generate a job j with $fix_j = \{u_j, v_j\}$, $j = 1, \dots, |A|$. We set $V_j = 2$ and $W_j = \frac{V_j}{|fix_j|} = 1$ for all $j = 1, \dots, n$. In the decision version of $P|fix_j, W_j = 1, energy|\sum C_j$ it is required to answer the question: Is there a schedule, in which the total completion time is not greater than a given threshold T?
In order to define the value of T, we solve an auxiliary problem with $2n$ single-processor jobs, i.e. each two processor job is replaced by two single-processor jobs. Such problem has the unique optimal solution (with the accuracy of permuting jobs on processors), where each processor executes three jobs and uses energy budget of 3. Now we find optimal durations of jobs on each processor, using the following convex model:

$$p_1 + 2p_2 + p_3 \to \min, \tag{5.1}$$

$$\sum_{j=1}^{3} p_j^{1-\alpha} = 3, \tag{5.2}$$

$$p_j \geq 0,\ j = 1, \dots, n. \tag{5.3}$$

Here p_j is the duration of j-th job on a processor.
We compose the Lagrangian function

$$L(p_j, \lambda) = (3p_1 + 2p_2 + p_3) + \lambda\left(p_1^{1-\alpha} + p_2^{1-\alpha} + p_3^{1-\alpha} - 3\right)$$

and calculate

$$p_j^* = \frac{3^{1/1-\alpha}}{(4-j)^{1/\alpha}}\left(3^{\alpha-1/\alpha} + 2^{\alpha-1/\alpha} + 1\right)^{1/\alpha-1},\ j = 1, 2, 3, \tag{5.4}$$

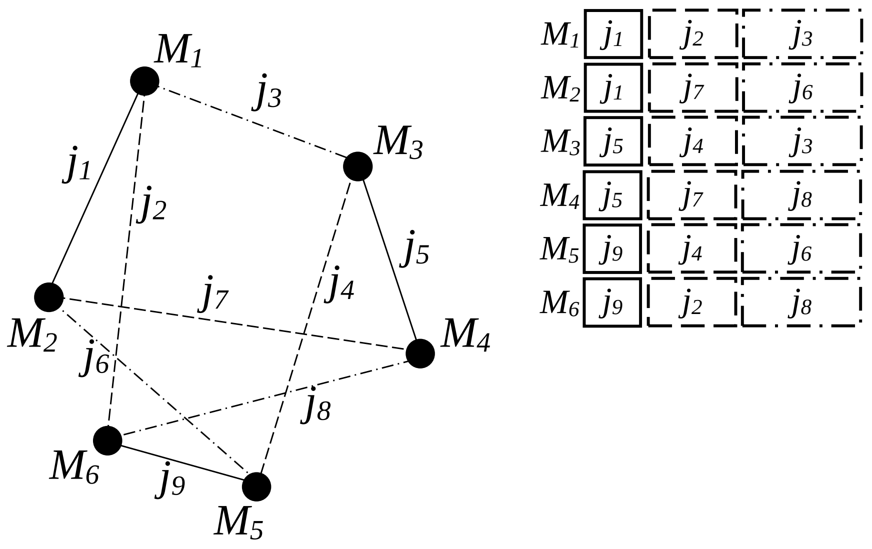

Figure 3: An example of the cubic graph with $\chi'(G) = 3$ (different color classes are marked by different types of lines) and the corresponding schedule of $P|fix_j, energy|\sum C_j$.

$$\sum C_j^* = \left(\frac{\left(3^{\alpha-1/\alpha} + 2^{\alpha-1/\alpha} + 1\right)^{\alpha}}{3} \right)^{1/\alpha - 1}. \qquad (5.5)$$

Here we apply KKT conditions as in the proof of Theorem 3.1. The sum of job completion times on all processors is equal to $m\sum C_j^*$. The optimal schedule does not have idle times. Set the threshold $T := \frac{m}{2}\sum C_j^*$ because at most $\frac{m}{2}$ two processor jobs can be executed simultaneously. We prove that a positive answer to the Chromatic Index problem corresponds to a positive one to the constructed decision version of $P|fix_j, W_j = 1, energy|\sum C_j$ and otherwise.

Now we assume that the answer to the Chromatic Index problem is positive. Then there is a feasible schedule, where $\frac{m}{2}$ jobs, corresponding to $\frac{m}{2}$ edges forming one color class, are executed in parallel (see example in Fig. 3). This schedule is similar to the optimal schedule of the corresponding problem with single-processor jobs. The value of criterion is equal to $\frac{m}{2}\sum C_j^*$.

We now show that if the sum of job completion times is no greater than $T = \frac{m}{2}\sum C_j^*$, then the corresponding schedule of jobs must constitute an affirmative answer to the Chromatic Index problem. Note that each processor must execute 3 jobs of processing work equal to 1. So, the minimum sum of job completion times on each processor is equal to (5.5) and it is reached, when the energy budget of $3|V|$ is distributed uniformly between processors. In this case, the schedule for two-processor jobs is identical to the optimal schedule for the corresponding single-processor jobs due to threshold T. Therefore, three subsets of $\frac{m}{2}$ jobs executed simultaneously give three color classes for the Chromatic Index problem.

The presented reduction is polynomial. So, speed scaling scheduling problem $P|fix_j, W_j = 1, energy|\sum C_j$ is strongly NP-hard. $\square$

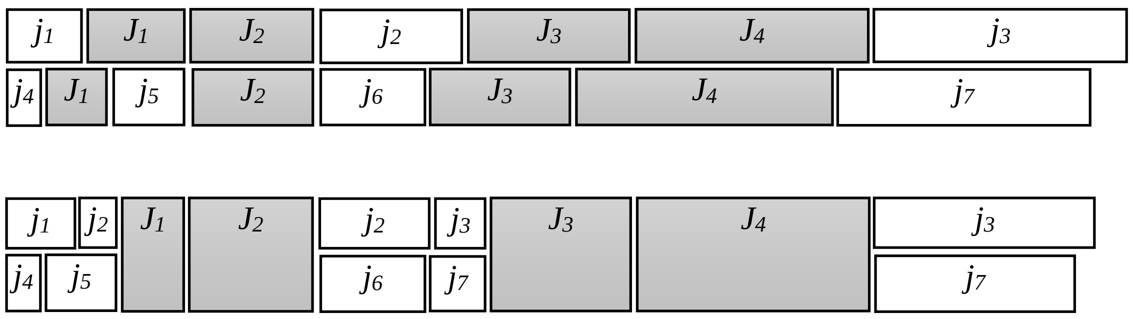

Figure 4: An example of single-processor schedules and the resulting preemptive two-processor schedule constructed by algorithm $\mathcal{A}^{pr}$.

5.2 Two-processor Instances

Here a constant-factor approximation algorithm is presented for the two-processor problem (denoted by P). Let $\mathcal{J}_i$ be the subset of jobs, that requires only processor $i = 1, 2$, and $\mathcal{J}_{12}$ is the subset of jobs with $fix_j = \{1, 2\}$, i.e. $\mathcal{J} = \mathcal{J}_1 \cup \mathcal{J}_2 \cup \mathcal{J}_{12}$. We identify two subproblems: the first subproblem P_1 is induced by jobs $\mathcal{J}' = \mathcal{J}_1 \cup \mathcal{J}_{12},\ |J'| = n'$, and the second subproblem P_2 contains only jobs $\mathcal{J}'' = \mathcal{J}_2 \cup \mathcal{J}_{12},\ |J''| = n''$.

The optimum $\sum C_j^1$ $(\sum C_j^2)$ of P_1 (P_2) is reached on the permutation, where the jobs are ordered by non-decreasing of values $W_i|fix_i|^{1/\alpha},\ i \in \mathcal{J}'$ $(W_i|fix_i|^{1/\alpha},\ i \in \mathcal{J}'')$, see Subsection 6.3. The value $\max\{\sum C_j^1,\ \sum C_j^2\}$ is the lower bound, say $LB(P)$, for the general problem P.

Decreasing the energy budget in both subproblems two times, we obtain $2^{1/\alpha-1}$-approximate solutions S' and S'' for them with the same sequences of jobs as in the case of energy budget E (see formula (6.5) in Section 6). Let C_j' and C_j'' (p_j' and p_j'') denote the completion times (the durations) of job j in S' and S'', respectively. Note that subsequences of two-processor jobs are identical in optimal solutions of both subproblems. Now we construct a preemptive schedule for the general problem and then transform this schedule to the non-preemptive one.

Approximation Algorithm $\mathcal{A}^{pr}$ for Preemptive Scheduling.

1. We non-preemptively schedule two-processor jobs j in time intervals $(\max\{C_j', C_j''\} - \min\{p_j', p_j''\},\ \max\{C_j', C_j''\}]$.

2. Single-processor jobs are executed without idle times in the same sequence as in S' and S'' (durations are not changed). Two-processor jobs may preempt single-processor ones.

Note that execution intervals of two-processor jobs do not intersect each other, therefore, the constructed preemptive schedule is feasible (see example in Fig. 4). The total completion time $\sum C_j(\mathcal{A}^{pr}) \leq 2 \cdot 2^{1/\alpha-1} LB(P)$ as the completion times of jobs are no later than in S' and S'' by construction.

Now we go to calculate a non-preemptive feasible schedule. The obtained preemptive schedule may be reconstructed without increasing the completion times of jobs such that at most

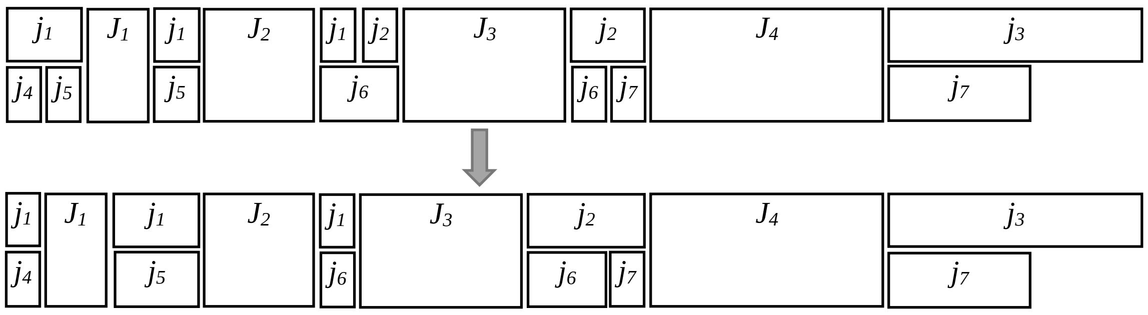

Figure 5: Reconstruction of the preemptive schedule.

one single-processor job is preempted by each two-processor job (if a two-processor job j preempts two single-processor jobs i and i', then changing the starting time of j to the maximum of starting times of i and i' and moving the corresponding parts of i and i' later will lower the completion time of j without affecting the completion times of any other jobs, see Fig. 5). Let $S'(j)$ and $C'(j)$ denote the starting time and completion time, respectively, of any job j in the reconstructed preemptive schedule.

Approximation Algorithm $\mathcal{A}^{npr}$ for Non-Preemptive Scheduling.
1. Identify single-processor jobs $j_{i_1},\ j_{i_2},\dots,j_{i_k}$ that are preempted by some two-processor jobs. Suppose that these jobs are ordered by increasing of starting times $S'(j_{i_1}) < \cdots < S'(j_{i_k})$, and therefore completion times satisfy $C'(j_{i_1}) < \cdots < C'(j_{i_k})$.
2. The main construction procedure consists of k steps.
At step l we move job j_{i_l}. Let $F(j_{i_l})$ be the last two-processor job that preempts j_{i_l}, $g(j_{i_l})$ be the amount of duration of j_{i_l} scheduled before the starting time of $F(j_{i_l})$, $h(j_{i_l})$ be the number of jobs that complete later than $F(j_{i_l})$, $l = 1,\dots,k$. Change the start time of j_{i_l} to the completion time of $F(j_{i_l})$ and increase starting times of the subsequent jobs by $g(j_{i_l})$.

Lemma 5.2. *Algorithm $\mathcal{A}^{npr}$ generates a feasible schedule with the total completion time $\sum C_j(\mathcal{A}^{npr}) \le 2\sum C_j(\mathcal{A}^{pr}) \le 2^{2\alpha-1/\alpha-1}LB(P)$ for problem $P2|fix_j, energy|\sum C_j$.*

Proof. At step l (see Fig. 6), inserting the idle time period will increase the total completion time of the schedule by $g(j_{i_l}) \cdot h(j_{i_l})$, and in total after all steps the non-preemptive schedule has the objective value

$$\sum C_j(\mathcal{A}^{npr}) \le \sum C_j(\mathcal{A}^{pr}) + \sum_{l=1}^{k} g(j_{i_l}) \cdot h(j_{i_l}).$$

Since each two-processor job preempts at most one single-processor job, the time intervals $(S'(j_{i_1}), C'(F(j_{i_1})]$, $(S'(j_{i_2}), C'(F(j_{i_2})]$,..., $(S'(j_{i_k}), C'(F(j_{i_k})]$ do not overlap with each other. Therefore,

$$\sum C_j(\mathcal{A}^{pr}) \ge \sum_{l=1}^{k} \big(C'(F(j_{i_l})) - S'(j_{i_l})\big) \cdot h(j_{i_l}) > \sum_{l=1}^{k} g(j_{i_l}) \cdot h(j_{i_l}).$$

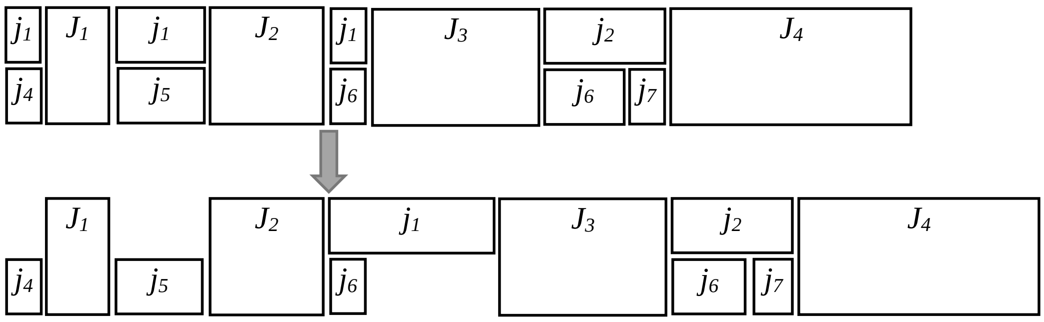

Figure 6: Step $l = 1$ in approximation algorithm $\mathcal{A}^{npr}$.

As a result, we have the following bound on the total completion time

$$\sum C_j(\mathcal{A}^{npr}) \le \sum C_j(\mathcal{A}^{pr}) + \sum_{l=1}^{k} g(j_{i_l}) \cdot h(j_{i_l}) < 2 \sum C_j(\mathcal{A}^{pr}) \le 2^{2\alpha-1/\alpha-1} LB(P).$$

$\square$

Theorem 5.3. *A $2^{2\alpha-1/\alpha-1}$-approximate schedule can be found in polynomial time for problem $P2|fix_j, energy|\sum C_j$.*

Theorem 5.4. *A $2^{\alpha/\alpha-1}$-approximate schedule can be found in polynomial time for problem $P2|fix_j, pmtn, energy|\sum C_j$.*

The complexity status of both preemptive and non-preemptive problems with two processors is open.

6 Polynomially Solvable Cases

In this section we provide a polynomial time algorithm for the easy particular cases, when there is an optimal solution, in which no more than one job is executed at each time moment.
Consider the following convex program in the case when all jobs are executed sequentially in according to permutation π:

$$F(\pi) = \sum_{i=1}^{n} (n - i + 1) p_{\pi_i} \to \min, \tag{6.1}$$

$$\sum_{i=1}^{n} (B_{\pi_i})^{\alpha} p_{\pi_i}^{1-\alpha} = E, \tag{6.2}$$

$$p_j \ge 0, \ j \in \mathcal{J}. \tag{6.3}$$

Here B_{π_i} is some parameter of job π_i, including the volume and the number of utilized processors.
In order to find an optimal solution of program (6.1)-(6.3) we use the Lagrangian method constructing the Lagrangian function

$$L(p_j,\lambda)=\sum_{i=1}^{n}(n-i+1)p_{\pi_i}+\lambda\left(\sum_{i=1}^{n}(B_{\pi_i})^{\alpha}p_{\pi_i}^{1-\alpha}-E\right).$$

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

$$p_{\pi_i}=\frac{B_{\pi_i}}{(n-i+1)^{\frac{1}{\alpha}}}\left(\sum_{j=1}^{n}B_{\pi_j}(n-j+1)^{\frac{\alpha-1}{\alpha}}\right)^{\frac{1}{\alpha-1}}(E)^{\frac{1}{1-\alpha}}, i=1,\dots,n. \tag{6.4}$$

So, the optimal objective for the given sequence π

$$F^*(\pi)=\left(\sum_{i=1}^{n}B_{\pi_i}(n-i+1)^{\frac{\alpha-1}{\alpha}}\right)^{\frac{\alpha}{\alpha-1}}(E)^{\frac{1}{1-\alpha}}. \tag{6.5}$$

The minimization of (6.5) is equivalent to the minimization of

$$f(\pi)=\sum_{i=1}^{n}B_{\pi_i}(n-i+1)^{\frac{\alpha-1}{\alpha}}$$

over all possible permutations π. We define n-dimensional vectors $B_\pi=(B_{\pi_1},B_{\pi_2},\dots,B_{\pi_n})$ and $N=\left(n^{\frac{\alpha-1}{\alpha}},(n-1)^{\frac{\alpha-1}{\alpha}},\dots,1^{\frac{\alpha-1}{\alpha}}\right)$. Then $f(\pi)$ can be expressed as the following scalar product $B_\pi\bullet(N)^{\mathrm{T}}$. It is easy to see that the minimum of $B_\pi\bullet(N)^{\mathrm{T}}$ is reached on the permutation, where the jobs are ordered by non-decreasing of values $B_j,\ j\in\mathcal{J}$. Indeed, in this case, the first vector is ordered nondecreasingly, the second one is ordered decreasingly, and the scalar product is minimal.

6.1 Single-processor Problem

The single-processor problem can be formulated as the following convex program in the case when a jobs sequence π is given:

$$\sum_{j\in\mathcal{J}}C_j(\pi)=\sum_{i=1}^{n}(n-i+1)p_{\pi_i}\to\min,$$

$$\sum_{i=1}^{n}(W_{\pi_i})^{\alpha}p_{\pi_i}^{1-\alpha}=E,$$

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

So, the minimum sum of completion times is reached on the permutation, where the jobs are ordered by non-decreasing volumes $W_j,\ j\in\mathcal{J}$.

Theorem 6.1. *An optimal schedule can be found in polynomial time for problem* $1|energy|\sum C_j$.

6.2 Fully Parallelizable Jobs

Now we show polynomial solvability of the problem with malleable jobs in the case when all jobs may use up to m processors (so called fully parallelizable jobs). We consider the linear case, when the runtime of a job decreases linearly with the number of processors assigned to it.

Obviously, there is an optimal schedule where each job utilizes m processors (see, e.g., (Turek et al., 1994)). Such problem is identical to the problem of minimizing the sum of completion times of jobs on single processor. Let jobs are ordered in accordance with permutation π and $W_j = \frac{V_j}{m}$, then we have the following convex problem

$$\sum_{i=1}^{n}\sum_{j=1}^{i} p_{\pi_j} = \sum_{i=1}^{n}(n-i+1)p_{\pi_i} \to \min,$$

$$\sum_{i=1}^{n} W_i^{\alpha} p_i^{1-\alpha} \le \frac{E}{m},$$

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

So, the minimum sum of completion times is reached on the permutation, where the jobs are ordered by non-decreasing of the required works $W_i,\ i \in \mathcal{J}$.

Theorem 6.2. *An optimal schedule can be found in polynomial time for problem $P|var, \delta_j = m, energy|\sum C_j$.*

6.3 Incompatible Jobs

The jobs are called incompatible when no two jobs can be executed simultaneously. In the case of single mode jobs this means that $fix_j \cap fix_i \neq \emptyset$ for all $i \neq j \in \mathcal{J}$. Rigid jobs are incompatible, when all $size_j > \frac{m}{2},\ j \in \mathcal{J}$. Such problem is also similar to the single-processor problem, since at most one job can be performed at each time moment. Suppose that jobs are ordered in accordance with some permutation π. In order to find optimal durations of jobs and the objective function for permutation π we solve the following problem:

$$\sum_{j=1}^{n} p_{\pi_j}(n-j+1) \to \min,$$

$$\sum_{j=1}^{n} m_j p_j \left(\frac{W_j}{p_j}\right)^{\alpha} = E,$$

$$p_j \ge 0,\ j = 1, \dots, n.$$

Here $m_j = |fix_j|$ for single mode jobs and $m_j = size_j$ for rigid jobs. The optimal schedule can be constructed from the optimal solution of the presented model corresponding to the permutation, where jobs are sequenced in the order of non-decreasing $W_j m_j^{1/\alpha}$.

Theorem 6.3. *Problem $P|fix_j, energy|\sum C_j$ is polynomially solvable in the case, when $fix_j \cap fix_i \neq \emptyset$ for all $i \neq j \in \mathcal{J}$.*

Theorem 6.4. *Problem $P|size_j, energy|\sum C_j$ is polynomially solvable in the case, when $size_j > \frac{m}{2}$ for all $j \in \mathcal{J}$.*

Conclusion

We provide the NP-hardness proofs for scheduling problems with the total completion time criterion and limited energy consumption. The specific property of the considered problems is resource-dependent durations of jobs. A method for designing of approximation algorithms is presented. Here a permutation of jobs and their speeds are identified, and then a schedule is formed by a greedy technique. The method allows us to obtain approximation guarantees for instances with rigid and moldable jobs, as well as two-processor cases.
Further research may be undertaken to experimental evaluation of the proposed algorithms. Open questions are the complexity status of the problem with moldable jobs and two-processor problems, and a constant factor approximation guarantee for two-processor jobs in the system with arbitrary number of processors. Moreover, it is interesting to investigate generalizations of the approach to heterogeneous models of speed scaling and alternative modes of jobs.

Acknowledgment

This work was supported by Russian Science Foundation grant 21-41-09017.

References

Albers, S. and Fujiwara, H. 2007. Energy-efficient algorithms for flow time minimization, *DACM Trans. Algorithms* **3**(4): 17 p.

Angel, E., Bampis, E., Kacem, F. and Letsios, D. 2012. Speed scaling on parallel processors with migration, *Euro-Par 2012 Parallel Processing*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 128–140.

Bampis, E., Letsios, D. and Lucarelli, G. 2014. A note on multiprocessor speed scaling with precedence constraints, *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures (SPAA-14)*, pp. 138–142.

Bampis, E., Letsios, D., Milis, I. and Zois, G. 2016. Speed scaling for maximum lateness, *Theory Comput Syst* **58**: 304–321.

Bansal, N. and Pruhs, K. 2005. Speed scaling to manage temperature, *STACS 2005*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 460–471.

Bansal, N., Pruhs, K. and Stein, C. 2009. Speed scaling for weighted flow time, *SIAM J. COMPUT* **39**(4): 1294–1308.

Boyd, S. and Vandenberghe, L. 2004. *Convex Optimization*, Cambridge University Press.

Bunde, D. 2009. Power-aware scheduling for makespan and flow, *J. Sched.* **12**: 489–500.

Cai, X., Lee, C.-Y. and Li, C.-L. 1998. Minimizing total completion time in two-processor task systems with prespecified processor allocations, *Naval Research Logistics* **45**(2): 231–242.

Drozdowski, M. 2009. *Scheduling for Parallel Processing*, Springer-Verlag, London.

Drozdowski, M. and Dell'Olmo, P. 2000. Scheduling multiprocessor tasks for mean flow time criterion, *Computers and Operations Research* **27**(6): 571–585.

Giaro, K., Kubale, M., Maiafiejski, M. and Piwakowski, K. 1999. Chromatic scheduling of dedicated 2-processor UET tasks to minimize mean flow time, *Proceedings of 7th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA–99)*, pp. 343–347.

Holyer, I. 1981. The NP-completeness of edge-coloring, *SIAM Journal on Computing* **10**: 718–720.

Hoogeveen, J., van de Velde, S. and Veltman, B. 1994. Complexity of scheduling multiprocessor tasks with prespecified processor allocations, *Discrete Applied Mathematics* **55**(3): 259–272.

Kononov, A. and Kovalenko, Y. 2020. Makespan minimization for parallel jobs with energy constraint, *Mathematical Optimization Theory and Operations Research, MOTOR-2020, LNCS*, Vol. 12095, Springer, Cham.

Kubale, M. 1996. Preemptive versus nonpreemtive scheduling of biprocessor tasks on dedicated processors, *European Journal of Operational Research* **94**(2): 242–251.

Kuhn, H. and Tucker, A. 1951. Nonlinear programming, *The Second Berkeley Symposium on Mathematical Statistics and Probability*, University of California Press, Berkeley, CA, pp. 481–492.

Lee, C.-Y. and Cai, X. 1999. Scheduling one and two-processor tasks on two parallel processors, *IEE Transactions* **31**(5): 445–455.

Pruhs, K., Uthaisombut, P. and Woeginger, G. 2008a. Getting the best response for your erg, *ACM Trans. Algorithms* **4**(3): 17 p.

Pruhs, K., Uthaisombut, P. and Woeginger, G. 2008b. Getting the best response for your erg, *ACM Trans. Algorithms.* **4**(3): 1–17.

Schwiegelshohn, U., Ludwig, W., Wolf, J., Turek, J. and Yu, P. 1998. Smart SMART bounds for weighted response time scheduling, *SIAM Journal on Computing* **28**: 237–253.

Shabtay, D. and Kaspi, M. 2006. Parallel machine scheduling with a convex resource consumption function, *Eur. J. Oper. Res.* **173**: 92–107.

Turek, J., Ludwig, W., Wolf, J., Fleischer, L., Tiwari, P., Glasgow, J., Schwiegelshohn, U. and Yu, P. 1994. Scheduling parallelizable tasks to minimize average response time, *Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures*, pp. 200–209.