Noname manuscript No.
(will be inserted by the editor)

Speed Scaling on Parallel Processors with Migration

Eric Angel · Evripidis Bampis · Fadi Kacem · Dimitrios Letsios

Received: date / Accepted: date

Abstract We study the problem of scheduling a set of jobs with release dates, deadlines and processing requirements (or works) on parallel speed scalable processors so as to minimize the total energy consumption. We consider that both preemptions and migrations of jobs are allowed. For this problem, there exists an optimal polynomial-time algorithm which uses as a black box an algorithm for linear programming. Here, we formulate the problem as a convex program and we propose a combinatorial polynomial-time algorithm which is based on finding maximum flows. Our algorithm runs in $O(nf(n)\log U)$ time, where n is the number of jobs, U is the range of all possible values of processors' speeds divided by the desired accuracy and $f(n)$ is the time needed for computing a maximum flow in a layered graph with $O(n)$ vertices.

Keywords Energy efficient scheduling · Speed scaling · Network flows · Convex programming

Work supported by (i) the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010, (ii) GDR du CNRS, RO, (iii) the European Research Council, Grant Agreement No. 691672, and (iv) the German Research Foundation, project AL464/9-1.

E. Angel
IBISC, Université d'Évry Val d'Essonne, France
E-mail: angel@ibisc.fr

E. Bampis
Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, F-75005 Paris, France
E-mail: evripidis.bampis@lip6.fr

F. Kacem
Carthage University, Tunisia
E-mail: fadikacem@gmail.com

D. Letsios
Lehrstuhl Theoretische Informatik, Fakultät für Informatik, Technische Universität München, Germany
letsios@in.tum.de

1 Introduction

Energy consumption is a major issue in our days. Great efforts are devoted to the reduction of energy dissipation in computing environments ranging from small portable devices to large data centers. From an algorithmic point of view, new challenging optimization problems are studied in which the energy consumption is taken into account as a constraint or as the optimization goal itself (for recent reviews see [2], [3]). This later approach has been adopted in the seminal paper of Yao et al. [22] who considered the problem of scheduling a set of jobs with release dates and deadlines on a single processor so that the total energy is minimized, under the so-called *speed scaling* model in which the speed of a processor can be varied over time and the power consumption is a convex function of the processor's speed. Specifically, at a given time t, if the speed of a processor is $s(t)$, then the power consumption is $s(t)^\alpha$, where $\alpha > 1$ is a constant, and the energy consumption is the power integrated over time.

Single processor Yao et al. [22] proposed an optimal offline algorithm, known as the YDS algorithm according to the initials of the authors, for the problem of minimizing the energy with preemptions, i.e. where the execution of a job may be interrupted and resumed later on. In the same work, they initiated the study of online algorithms for this problem, introducing the Average Rate (AVR) and the Optimal Available (OA) algorithms. Bansal et al. [8] proposed a new online algorithm, the BKP algorithm according to the authors' initials, which has better competitive ratio than AVR and OA for large values of α.

Multiprocessor When there are multiple processors available, we may consider different variants of speed scaling problems. In the *non-migratory* variant, we allow preemptions of jobs but not migrations. This means that a job may be interrupted and resumed later on the same processor, but it is not allowed to continue its execution on a different processor. In the *migratory* variant, both preemptions and migrations of jobs are allowed.

Albers et al. [6] considered the multiprocessor non-migratory problem of minimizing the energy of a set of unit-work jobs. For the case in which the jobs have *agreeable* deadlines, they proposed an optimal polynomial time algorithm. Then, they showed that the problem is $\mathcal{NP}$-hard when the jobs have arbitrary deadlines and they proposed an $\alpha^\alpha 2^{4\alpha}$-approximation algorithm for it. Note that, the ratio of this algorithm is constant because α is a constant. For the more general problem in which the jobs have arbitrary works, Greiner et al. [15] established a generic reduction transforming a ρ-approximation algorithm for the single-processor problem to a $\rho B_{\lceil\alpha\rceil}$-approximation algorithm for the multiprocessor non-migratory problem, where $B_{\lceil\alpha\rceil}$ is the $\lceil\alpha\rceil$-th Bell number. This result basically implies a constant factor approximation algorithm for the multiprocessor non-migratory problem with arbitrary works.

For the migratory variant, Chen et al. [13] initiated the study of the multiprocessor speed scaling problem of minimizing the energy and they proposed a greedy algorithm for the basic setting in which all jobs have the same release

date and deadline. Bingham and Greenstreet [11] proposed a polynomial-time algorithm for the more general setting where the jobs have arbitrary release dates and deadlines. The algorithm in [11] needs an algorithm for linear programming as a black box. So, after this work, an intriguing open question was whether there exists a faster combinatorial algorithm.

It has to be noticed that after the conference version of our paper, multiprocessor speed scaling problems attracted much attention and new papers appeared (e.g. [9] and [7]) in the direction of improving the above results and generalizing them for power-heterogeneous environments in which different processors obey to different power functions.

Multi-objective In general, energy and performance are conflicting objectives and a series of papers addressed speed scaling problems in a multicriteria context. Pruhs et al. [20] studied the problem of optimizing performance without exceeding a fixed budget of energy. Their objective was total flow time minimization and they presented an optimal polynomial-time algorithm for instances with unit-work jobs. In order to prove the optimality of their algorithm, they formulated the problem as a convex program and they showed that their algorithms always produces a solution satisfying the well-known Karush, Kuhn, Tucker (KKT) conditions which are necessary and sufficient conditions for optimality in convex programming. Albers and Fujiwara [5] studied the problem of minimizing the total flow time plus energy which is an alternative way for optimizing two conflicting objectives. Finally, Chan et al. [12] proposed an online algorithm for maximizing the throughput and minimizing the energy of a set of jobs which have to be executed by a processor whose speed is bounded above. Their algorithm has constant competitive ratio in terms of both objectives.

Our contribution and organization of the paper We consider the multiprocessor migratory speed scaling problem and our objective is energy minimization. In Section 3, we formulate the problem as a convex program. In Section 4, we apply the KKT conditions to our convex program and we obtain a set of structural properties of optimal schedules. Then, in Section 5, we propose an optimal algorithm which is based on maximum flow computations. The running time of our algorithm, which we call BAL, is $O(nf(n)\log U)$, where n is the number of jobs, U is the range of all possible values of processors' speed divided by the desired accuracy and $f(n)$ is the complexity of computing a maximum flow in a layered graph with $O(n)$ vertices. Independently to our work, Albers et al. [4] proposed another optimal algorithm for the same problem which also explores the relation of the problem with maximum flow.

2 Preliminaries

We begin with a formal description of our problem and some definitions. An instance of our problem consists of a set of jobs $\mathcal{J} = \{J_1, ..., J_n\}$ which have to

be executed by m parallel processors. Each job J_j is specified by an amount of work w_j, a release date r_j and a deadline d_j. The processors are *speed scalable* which means that they can vary their speeds dynamically over time. Every processor satisfies the speed-to-power function $P(s) = s^\alpha$, where $P(s)$ is the processor's power consumption if it runs with speed s and $\alpha > 1$ is a constant. It has to be noticed that the speed of a processor may be any non-negative value. If a processor runs with speed s for t units of time, then it executes $t \cdot s$ units of work and it consumes $t \cdot s^\alpha$ units of energy. In our setting, we allow preemptions and migrations of the jobs. That is, a job may be executed on some processor, suspended and resumed later on the same or on a different processor. However, we do not allow parallel execution of a job which means that a job cannot be run by more that one processors at a given time. Our objective is to find a schedule with minimum energy consumption so that every job is entirely executed between its release date and its deadline.

An important remark is that there is always an optimal schedule for our problem such that every job J_j is executed with fixed speed s_j. This is a well-known fact for most speed scaling problems (see [22]) and it can be derived by using the convexity of the speed-to-power function.

Let $t_0 < t_1 < \ldots < t_\ell$ be the times which correspond to all the release dates and deadlines of the jobs. We define the intervals $I_i = [t_{i-1}, t_i)$, for all $1 \leq i \leq \ell$, and we refer by $\mathcal{I}$ to the set of all these intervals, i.e. $\mathcal{I} = \{I_1, I_2, \ldots, I_\ell\}$. For a time interval $I_i \in \mathcal{I}$, we denote by $|I_i|$ its length and by $\mathcal{A}_i$ the set of jobs which are allowed to be executed during I_i, i.e. $\mathcal{A}_i = \{J_j \in \mathcal{J} : I_i \subseteq [r_j, d_j)\}$. If $J_j \in \mathcal{A}_i$, then we say that job J_j is *active* during I_i.

Next, we state a problem which we call *Work Assignment Problem* (WAP) and which is a key ingredient for our analysis. WAP resembles to our original problem except that the speed of each processor is fixed (it cannot be varied) and the number of available processors is not the same at each time. An instance of WAP consists of a set of jobs $\mathcal{J} = \{J_1, ..., J_n\}$ and a set of disjoint time intervals $\mathcal{I} = \{I_1, I_2, \cdots, I_\ell\}$. Each job J_j has an amount of work w_j and it is active (i.e. it is allowed to be executed) in a subset of the intervals. During interval I_i there are m_i available processors. Furthermore, we are given a value v. Our objective is to decide whether there exists a feasible schedule in which all jobs are feasibly executed with a fixed speed v. Note that a schedule is feasible only if each job is entirely executed during the intervals that it is active. Preemptions and migrations of jobs are allowed but parallel execution of a job is not permitted. In WAP, each job J_j has a fixed processing time equal to $\frac{w_j}{v}$. Therefore, WAP can be solved in polynomial time with a variant of the optimal algorithm of the well-known scheduling problem $P|pmtn, r_j, d_j|-$ (see [10]).

3 Convex Programming Formulation

In this section, we propose a convex program for our problem. We introduce the variables s_j and $t_{i,j}$, for all $J_j \in \mathcal{J}$ and $I_i \subseteq [r_j, d_j)$, corresponding to the

speed of job J_j and the total amount of time that job J_j is processed during I_i, respectively. Then, the problem can be formulated as follows:

$$\min \sum_{J_j \in \mathcal{J}} w_j s_j^{\alpha-1} \tag{1}$$

$$\sum_{I_i \subseteq [r_j, d_j)} t_{i,j} \geq \frac{w_j}{s_j} \qquad J_j \in \mathcal{J} \tag{2}$$

$$\sum_{J_j \in \mathcal{A}_i} t_{i,j} \leq m \cdot |I_i| \qquad I_i \in \mathcal{I} \tag{3}$$

$$t_{i,j} \leq |I_i| \qquad J_j \in \mathcal{J},\ I_i \subseteq [r_j, d_j) \tag{4}$$

$$t_{i,j} \geq 0 \qquad J_j \in \mathcal{J},\ I_i \subseteq [r_j, d_j) \tag{5}$$

$$s_j \geq 0 \qquad J_j \in \mathcal{J} \tag{6}$$

Expression (1) corresponds to the total energy consumption. Constraints (2) enforce that at least w_j units of work are executed for each job J_j. Because of the constraints (3), at most m processors are used at each time. Due to the constraints (4), a job J_j is executed by at most one processor at each time. Constraints (5) and (6) ensure the non-negativity of the variables s_j and $t_{i,j}$. It has to be mentioned that Constraints (1)-(6) are not sufficient for feasibility, i.e. there exist unfeasible schedules satisfying all of them. However, by applying the well-known McNaughton's algorithm [18] in each interval $I_i \in \mathcal{I}$, every such schedule can be converted into a feasible one without increasing the total energy consumption.

The single variable functions $w_j s_j^{\alpha-1}$ are convex, as $(w_j s_j^{\alpha-1})'' = (\alpha - 1)(\alpha - 2) w_j s_j^{\alpha-3} > 0$, for each $\alpha > 2$. Similarly, the single variable functions w_j/s_j are also convex. Because the objective function and Constraints (2) consist of convex function sums while all remaining constraints are linear, the above mathematical program is convex for $\alpha > 2$. By using variable t_j corresponding to job J_j processing time instead of variable s_j, we obtain an equivalent mathematical program which is convex for any $\alpha > 1$. In particular, every occurrence of s_j is replaced by w_j/t_j, the objective function becomes $\sum_{j \in \mathcal{J}} w_j^\alpha / t_j^{\alpha-1}$, Constraints (2) become $\sum_{I_i \subseteq [r_j, d_j)} t_{i,j} \geq t_j$, for $J_j \in \mathcal{J}$, and all other constraints remain identical. Function $w_j^\alpha / t_j^{\alpha-1}$ is convex, since $(w_j^\alpha / t_j^{\alpha-1})'' = \alpha(\alpha - 1) w_j^\alpha / t_j^{\alpha+1} > 0$, for each $\alpha > 1$. Because speed relationships are more intuitive, we elaborate on the convex program with the s_j variables. Nevertheless, our analysis may be equivalently applied using the convex program with the t_j variables and obtain an optimal algorithm for all $\alpha > 1$.

Since our problem can be formulated as a convex program, it can be solved in polynomial time as follows. We solve the convex program in polynomial time by applying the Ellipsoid Algorithm and we obtain a processing time (i.e. speed) for each job. Once these processing times have been computed, we construct a feasible schedule by using an optimal algorithm for the feasibility

problem $P|pmtn, r_j, d_j|-$. In the remainder of the paper, we derive a faster combinatorial algorithm.

4 Structure of an Optimal Schedule

In what follows, we elaborate on the structure of an optimal schedule for our problem and we derive some properties which are necessary and sufficient for optimality. These properties are obtained by using the well-known KKT (Karush, Kuhn, Tucker) conditions which are necessary and sufficient for optimality in convex programming (see [19]). A description of the KKT conditions can be found in the Appendix. We apply the KKT conditions to the convex program that we presented in the previous section and we obtain the following lemma which specifies the structure of an optimal schedule.

Lemma 1 *A feasible solution of the problem is optimal if and only if, for every $I_i \in \mathcal{I}$, the following properties hold:*

1. *If $|\mathcal{A}_i| \leq m$, then $t_{i,j} = |I_i|$ for every job $J_j \in \mathcal{A}_i$.*
2. *Otherwise, it holds that*
   - *i.* $\sum_{J_j \in \mathcal{A}_i} t_{i,j} = m \cdot |I_i|$.
   - *ii. All jobs $J_j \in \mathcal{A}_i$ with $0 < t_{i,j} < |I_i|$ have equal speeds.*
   - *iii. If a job $J_j \in \mathcal{A}_i$ is not executed during I_i, i.e. $t_{i,j} = 0$, then $s_j \leq s_{j'}$ for any job $J_{j'} \in \mathcal{A}_i$ with $t_{i,j'} > 0$.*
   - *iv. If a job $J_j \in \mathcal{A}_i$ is executed during the whole interval I_i, i.e. $t_{i,j} = |I_i|$, then $s_j \geq s_{j'}$ for any job $J_{j'} \in \mathcal{A}_i$ with $t_{i,j'} < |I_i|$.*

Proof The proof consists of two parts. Part 1 shows that a solution satisfying the KKT conditions also satisfies the lemma's properties. Part 2 shows that all feasible solutions satisfying the lemma's properties attain equal energy consumption and assign the same speeds to the jobs. Because there is always an optimal solution to the convex program satisfying the KKT conditions, the lemma follows.

Part 1 Consider an interval I_i s.t. $|\mathcal{A}_i| \leq m$. A job J_j cannot be executed for more than $|I_i|$ units of time during the interval I_i because we do not allow parallel execution of jobs. Assume for contradiction that there exists an optimal schedule $\mathcal{S}^*$ s.t. $t_{i,j} < |I_i|$, for some job $J_j \in \mathcal{A}_i$. Then, we can increase the time that J_j is processed during I_i in order to obtain a new schedule which is feasible because $|\mathcal{A}_i| \leq m$ and of lower energy consumption.

Now, we consider the more challenging case of an interval I_i s.t. $|\mathcal{A}_i| > m$. Note that, with a similar argument as before, we can show that $\sum_{J_j \in \mathcal{A}_i} t_{i,j} = m \cdot |I_i|$. We prove the remaining properties by applying the KKT conditions.

We associate to each constraint of the convex program a dual variable. So, to the constraints (2)-(5), we associate the dual variables λ_j, μ_i, $\pi_{i,j}$ and $\sigma_{i,j}$, respectively. Clearly, $s_j > 0$ for each job $J_j \in \mathcal{J}$. Therefore, by the

complementary slackness conditions, the dual variables associated with the constraints (6) must be equal to zero.

By stationarity conditions,

$$\nabla\left(\sum_{J_j\in\mathcal{J}} w_j s_j^{\alpha-1}\right) + \sum_{J_j\in\mathcal{J}} \lambda_j \cdot \nabla\left(\frac{w_j}{s_j} - \sum_{I_i\subseteq[r_j,d_j)} t_{i,j}\right) + \sum_{I_i\in\mathcal{I}} \mu_i \nabla\left(\sum_{J_j\in\mathcal{A}_i} t_{i,j} - m\cdot|I_i|\right) + \sum_{I_i\in\mathcal{I}}\sum_{J_j\in\mathcal{A}_i} \pi_{i,j}\nabla(t_{i,j}-|I_i|) + \sum_{I_i\in\mathcal{I}}\sum_{J_j\in\mathcal{A}_i} \sigma_{i,j}\nabla(-t_{i,j}) = 0 \Leftrightarrow$$

$$\sum_{J_j\in\mathcal{J}}\left((\alpha-1)w_j s_j^{\alpha-2} - \lambda_j\frac{w_j}{s_j^2}\right)\nabla s_j + \sum_{I_i\in\mathcal{I}}\sum_{J_j\in\mathcal{A}_i}\left(-\lambda_j+\mu_i+\pi_{i,j}-\sigma_{i,j}\right)\nabla t_{i,j} = 0$$

In order to satisfy the stationarity conditions, the coefficients of the partial derivatives ∇s_j and $\nabla t_{i,j}$ must be equal to zero. Thus, we get that

$$(\alpha-1)s_j^\alpha = \mu_i + \pi_{i,j} - \sigma_{i,j} \qquad I_i\in\mathcal{I},\ J_j\in\mathcal{A}_i \tag{7}$$

By complementary slackness conditions,

$$\pi_{i,j}\cdot(t_{i,j}-|I_i|) = 0 \qquad I_i\in\mathcal{I},\ J_j\in\mathcal{A}_i \tag{8}$$

$$\sigma_{i,j}\cdot(-t_{i,j}) = 0 \qquad I_i\in\mathcal{I},\ J_j\in\mathcal{A}_i \tag{9}$$

Note that, if we applied the KKT conditions in the formulation with processing time variables t_j instead of the speed variables s_j, then the would derive exactly the same conditions (7) - (9). For a given job $J_j\in\mathcal{A}_i$, we consider the following cases:

- $0 < t_{i,j} < |I_i|$
  Complementary slackness conditions (8) and (9) imply that $\pi_{i,j} = \sigma_{i,j} = 0$. As a result, (7) can be written as
  $$(\alpha-1)s_j^\alpha = \mu_i. \tag{10}$$
  Therefore, all such jobs have the same speed which is proportional to μ_i. That is, *2ii* is true.
- $t_{i,j} = 0$
  By (8), we have that $\pi_{i,j} = 0$ and (7) is expressed as $(\alpha-1)s_j^\alpha = \mu_i - \sigma_{i,j}$. Since $\sigma_{i,j}\geq 0$, we get that
  $$(\alpha-1)s_j^\alpha \leq \mu_i. \tag{11}$$
- $t_{i,j} = |I_i|$
  By (9), we get that $\sigma_{i,j} = 0$. So, (7) becomes $(\alpha-1)s_j^\alpha = \mu_i + \pi_{i,j}$. Given that $\pi_{i,j}\geq 0$,
  $$(\alpha-1)s_j^\alpha \geq \mu_i. \tag{12}$$

By equations (10), (11) and (12), we conclude that *2iii* and *2iv* are satisfied by an optimal schedule.

Part 2 Given an optimal solution of the convex program that satisfies the KKT conditions, we derived some relations between the primal variables. Based on them, we obtained some structural properties of an optimal schedule for our problem. Next, we show that these properties are also sufficient for optimality.

Assume for the sake of contradiction that there exists a non-optimal schedule $\mathcal{S}$ satisfying the properties of Lemma 1 and let $\mathcal{S}^*$ be an optimal schedule which satisfies Lemma 1. We denote by E, s_j and $t_{i,j}$ the energy consumption, the speed of job J_j and the total execution time of job J_j during I_i, respectively, in schedule $\mathcal{S}$. Let E^*, s_j^* and $t_{i,j}^*$ be the corresponding values for the schedule $\mathcal{S}^*$.

We define the set $\mathcal{J}'$ which contains the jobs J_j with $s_j > s_j^*$. Since $E > E^*$, $\mathcal{J}'$ is not empty. By the definition of $\mathcal{J}'$,

$$\sum_{J_j \in \mathcal{J}'} \sum_{I_i : \subseteq [r_j, d_j)} t_{i,j} < \sum_{J_j \in \mathcal{J}'} \sum_{I_i : \subseteq [r_j, d_j)} t_{i,j}^*$$

Hence, there is at least one interval I_i such that

$$\sum_{J_j \in \mathcal{J}'} t_{i,j} < \sum_{J_j \in \mathcal{J}'} t_{i,j}^*$$

If $|\mathcal{A}_i| \leq m$, then there is at least one job J_j such that $t_{i,j} < t_{i,j}^*$. Due to the property 1 of Lemma 1, it should hold that $t_{i,j} = t_{i,j}^* = |I_i|$ which is a contradiction. So, it must be the case that $|\mathcal{A}_i| > m$. Then, the last equation gives that $t_{i,j} < t_{i,j}^*$ for some job $J_j \in \mathcal{J}'$. Thus, $t_{i,j} < |I_i|$ and $t_{i,j}^* > 0$. Both schedules $\mathcal{S}$ and $\mathcal{S}^*$ must have equal sum of processing times during the interval I_i, by property 2i of Lemma 1. So, there must be a job $J_{j'} \notin \mathcal{J}'$ such that $t_{i,j'} > t_{i,j'}^*$. Therefore, $t_{i,j'} > 0$ and $t_{i,j'}^* < |I_i|$. We conclude that

$$s_{j'} \geq s_j > s_j^* \geq s_{j'}^*$$

The first inequality comes from the fact that $t_{i,j'} > 0$, $t_{i,j} < |I_i|$ and Lemma 1. The second inequality holds because $J_j \in \mathcal{J}'$. The third inequality is obtained similarly with the first inequality. Given the above, we have a contradiction on the fact that $J_{j'} \notin \mathcal{J}'$.

5 Optimal Combinatorial Algorithm

Next, we propose an optimal combinatorial algorithm for our problem which constructs schedules satisfying Lemma 1.

Our algorithm is based on the notion of *critical jobs* defined below. Initially, the algorithm conjectures that all jobs are executed at the same speed in the optimal schedule and it assigns to all of them a sufficiently large speed. The key idea is to continuously decrease the speeds of the jobs step by step. At each step, it assigns a speed to the critical jobs which are ignored in the subsequent steps and it goes on reducing the speeds of the remaining jobs. At the end of

the last step, every job has been assigned a speed. Critical jobs are recognized by finding a minimum (s,t)-cut in an appropriate graph as we describe in the following. Once the algorithm has computed a speed, i.e. a processing time, for each job, it constructs a feasible schedule by applying an optimal algorithm for $P|pmtn, r_j, d_j|-$.

At a given step, the algorithm performs a binary search in order to reduce the speeds of the jobs. The binary search is performed by solving repeatedly different instances of the WAP. Each instance of the WAP is solved by a maximum flow computation. Specifically, given an instance $< \mathcal{J}, \mathcal{I}, v >$ of the WAP, the algorithm constructs a directed graph G as follows. There is one node for each job $J_j \in \mathcal{J}$, one node for each interval $I_i \in \mathcal{I}$, a source node s and a destination node t. The algorithm introduces an arc (s, J_j), for each $J_j \in \mathcal{J}$, with capacity $\frac{w_j}{v}$, an arc (J_j, I_i) with capacity $|I_i|$, for each couple of job J_j and interval I_i such that $J_j \in \mathcal{A}_i$, and an arc (I_i, t) with capacity $m_i|I_i|$ for each interval $I_i \in \mathcal{I}$. We say that this is the *corresponding graph* of $< \mathcal{J}, \mathcal{I}, v >$. The algorithm decides if an instance $< \mathcal{J}, \mathcal{I}, v >$ of the WAP is feasible by computing a maximum (s,t)-flow on its corresponding graph G, based on the following theorem.

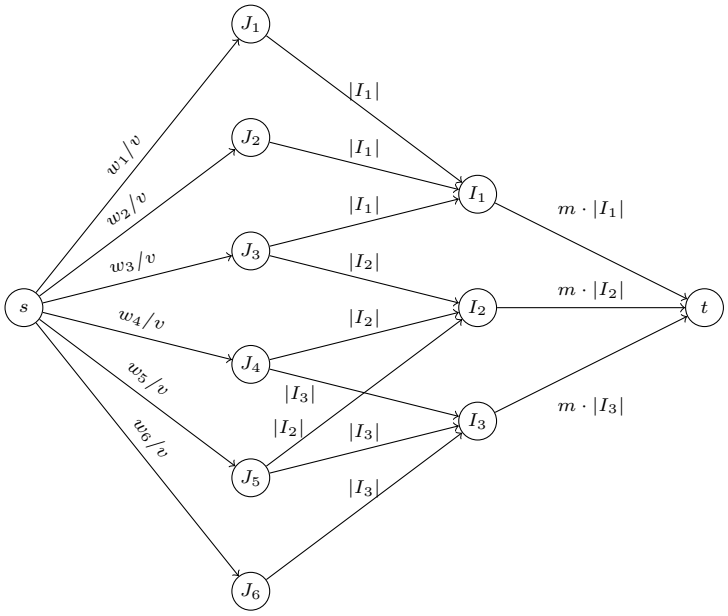

Fig. 1 Graph corresponding to an instance of the WAP.

Theorem 1 *There exists a feasible schedule for an instance* $< \mathcal{J}, \mathcal{I}, v >$ *of the WAP iff there exists a feasible* (s,t)*-flow of value* $\sum_{J_j \in \mathcal{J}} \frac{w_j}{v}$ *in the corresponding graph* G*.*

We are ready to introduce the notion of criticality for feasible instances of the WAP. Given a feasible instance for the WAP, we say that job J_c is *critical* if, for any feasible schedule $\mathcal{S}$ and for each interval I_i such that $J_c \in \mathcal{A}_i$, either $t_{i,c} = |I_i|$ or $\sum_{J_j \in \mathcal{A}_i} t_{i,j} = m_i |I_i|$, where $t_{i,j}$ is the total amount of time that job J_j is processed during I_i in $\mathcal{S}$. Moreover, we say that an instance $< \mathcal{J}, \mathcal{I}, v >$ of the WAP is critical if v is the minimum speed so that the set of jobs $\mathcal{J}$ can be feasibly executed during the intervals in $\mathcal{I}$. We refer to this speed v as the critical speed of $\mathcal{J}$ and $\mathcal{I}$.

Based on the Theorem 1, we extend the notion of criticality. Let us consider a feasible instance $< \mathcal{J}, \mathcal{I}, v >$ of the WAP and let G be its corresponding graph. Given an arc e and a feasible (s,t)-flow $\mathcal{F}$ of G, we say that the arc e is saturated by $\mathcal{F}$ if the amount of flow that crosses the arc e according to $\mathcal{F}$ is equal to the capacity of e. Additionally, we say that a path p of G is saturated by $\mathcal{F}$ if there exists at least one arc e in p which is saturated. Then, a job J_c is critical if every path J_c, I_i, t is saturated by any maximum (s,t)-flow $\mathcal{F}$.

In order to continue our analysis, we need the following lemma which relates, in a sense, the notions of *critical job* and *critical instance*.

Lemma 2 *If* $< \mathcal{J}, \mathcal{I}, v >$ *is a critical instance of WAP, then there is at least one critical job* $J_j \in \mathcal{J}$*.*

Proof Let G be the corresponding graph of $< \mathcal{J}, \mathcal{I}, v >$. Since the instance $< \mathcal{J}, \mathcal{I}, v >$ is critical, there exists a minimum (s,t)-cut $\mathcal{C}$ in G that contains either an arc (J_j, I_i), for some $J_j \in \mathcal{J}$ and $I_i \in \mathcal{I}$, or an arc (I_i, t), for some $I_i \in \mathcal{I}$. If this was not the case, the only minimum (s,t)-cut would be the one with all the arcs (s, J_j). This means that we could reduce the speed v to $v - \epsilon$, for an infinitesimal quantity $\epsilon > 0$, and the instance $< \mathcal{J}, \mathcal{I}, v - \epsilon >$ would admit a feasible flow equal to $\sum_{J_j \in \mathcal{J}} \frac{w_j}{v-\epsilon}$ which contradicts the criticality of $< \mathcal{J}, \mathcal{I}, v >$.

Now, there must be at least one arc (s, J_c) that does not belong to $\mathcal{C}$, which is a minimum (s,t)-cut containing at least one of the arcs (J_j, I_i) or (I_i, t). If all arcs (s, J_j) were included in $\mathcal{C}$, then $\mathcal{C}$ would have greater capacity than the (s,t)-cut that contains just all the arcs (s, J_j) in contradiction with the fact that $\mathcal{C}$ is a minimum (s,t)-cut. Based on the definition of an (s,t)-cut, we conclude that all paths J_c, I_i, t must have an arc that belongs in $\mathcal{C}$ so that if we remove the arcs of $\mathcal{C}$, the nodes s and t become disconnected. Hence, the job J_c is critical.

Note that the instance $< \mathcal{J}, \mathcal{I}, v - \epsilon >$ is not feasible if $< \mathcal{J}, \mathcal{I}, v >$ is critical. Up to now, the notion of a critical job has been defined only in the context of feasible instances. We extend this notion for unfeasible instances as follows. In an unfeasible instance $< \mathcal{J}, \mathcal{I}, v - \epsilon >$, a job J_j is called critical if every path J_j, I_i, t is saturated by any maximum (s,t)-flow in the corresponding graph G'.

Let $< \mathcal{J}, \mathcal{I}, v >$ be a critical instance of the WAP and let G be its corresponding graph. Next, we propose a way for identifying the critical jobs of $< \mathcal{J}, \mathcal{I}, v >$ using the graph G' that corresponds to the instance $< \mathcal{J}, \mathcal{I}, v-\epsilon >$, for some sufficiently small constant $\epsilon > 0$ based on Lemmas 3 and 4 below. The value of ϵ is such that the two instances have exactly the same set of critical jobs. Moreover, the critical jobs of $< \mathcal{J}, \mathcal{I}, v - \epsilon >$ can be found by computing a minimum (s,t)-cut in the graph that corresponds to $< \mathcal{J}, \mathcal{I}, v - \epsilon >$.

Lemma 3 *Given a critical instance $< \mathcal{J}, \mathcal{I}, v >$ of the WAP, there exists a sufficiently small constant $\epsilon > 0$ such that the unfeasible instance $< \mathcal{J}, \mathcal{I}, v - \epsilon >$ and $< \mathcal{J}, \mathcal{I}, v >$ have exactly the same critical jobs. The same holds for any other unfeasible instance $< \mathcal{J}, \mathcal{I}, v - \epsilon' >$ such that $0 < \epsilon' \leq \epsilon$.*

Proof Since $< \mathcal{J}, \mathcal{I}, v >$ is a critical instance, because of Lemma 2, it must contain at least one critical job.

If all the jobs of the instance are critical, then, in the graph G that corresponds to $< \mathcal{J}, \mathcal{I}, v >$, there is a minimum (s,t)-cut $\mathcal{C}$ that contains exactly one arc of every path J_j, I_i, t. Clearly, $\mathcal{C}$ is a minimum (s,t)-cut for the graph G' that corresponds to $< \mathcal{J}, \mathcal{I}, v - \epsilon >$ for any $\epsilon > 0$, because all the arcs (s, J_j), $J_j \in \mathcal{J}$, have greater capacity in G' than in G, while all the other arcs have equal capacities in the two graphs. Hence, for any job $J_j \in \mathcal{J}$, either the arc (J_j, I_i) or the arc (I_i, t) is saturated by any maximum (s,t)-flow in G', for all $I_i \in \mathcal{I}$ such that $J_j \in \mathcal{A}_i$. That is, all jobs are critical in G' as well and the lemma is true.

Now, assume that there is at least one non-critical job. Consider a non-critical job J_j. We know that there must be at least one maximum (s,t)-flow $\mathcal{F}$ in G such that at least one path J_j, I_i, t is not saturated by $\mathcal{F}$, for some $I_i \in \mathcal{I}$ such that $J_j \in \mathcal{A}_i$. Consider such a path J_j, I_i, t. Since the path is not saturated, we have that $c_{(J_j,I_i)} - f_{(J_j,I_i)} > 0$ and $c_{(I_i,t)} - f_{(I_i,t)} > 0$, where c_e is the capacity of the arc e and f_e is the amount of flow that passes through e according to $\mathcal{F}$, respectively. Then, we set

$$\eta_j = \min\{c_{(J_j,I_i)} - f_{(J_j,I_i)}, c_{(I_i,t)} - f_{(I_i,t)}\}$$

The intuition behind the value η_j is the following. Assume that we increase the capacity of the arc (s, J_j) while keeping the same capacities for the remaining arcs. If this increase is less than η_j, then there is a maximum (s,t)-flow $\mathcal{F}'$ in the new graph such that neither the arc (J_j, I_i), nor the arc (I_i, t) are saturated by $\mathcal{F}'$. The maximum (s,t)-flow $\mathcal{F}'$ in the new graph can be easily obtained from the maximum (s,t)-flow $\mathcal{F}$ in G.

For every non-critical job J_j, we fix a positive value η_j as we described in the previous paragraph. Note that we do not want to compute such a value but we only care for its existence. Let η_{min} be the minimum value η_j, among all the non-critical jobs. From the instance $< \mathcal{J}, \mathcal{I}, v >$, we obtain an unfeasible instance $< \mathcal{J}, \mathcal{I}, v - \epsilon >$ as follows. We pick an ϵ such that the total increase of the capacities of all the arcs from the source node to the job nodes is less

than η_{min}. In other words, the value ϵ must satisfy the following inequality

$$\sum_{J_j \in \mathcal{J}} \frac{w_j}{v-\epsilon} < \sum_{J_j \in \mathcal{J}} \frac{w_j}{v} + \eta_{min}$$

Let us, now, explain why the two instances have the same critical jobs. Initially, we will show that if a job is non-critical in G, then it remains non-critical in G'. By the way we picked ϵ, for any non-critical job J_j in G, there is always a maximum (s,t)-flow such that some path from J_j to t is not saturated in G'. Therefore, each non-critical job in G, remains a non-critical job in G'.

Next, consider a critical job J_j of $< \mathcal{J}, \mathcal{I}, v >$. By construction, the arc (s, J_j) has greater capacities in G' than in G and all the arcs (J_j, I_i) and (I_i, t), $J_j \in \mathcal{A}_i$, have equal capacities in the two graphs. We conclude that (s, J_j) cannot belong to any minimum (s,t)-cut in G'. Thus, every path J_j, I_i, t is saturated by any maximum (s,t)-flow in G'. Therefore, if a job is critical in G, then it is critical in G' as well.

The following lemma is a direct consequence of the definition of criticality on feasible and infeasible WAP instances.

Lemma 4 *Assume that $< \mathcal{J}, \mathcal{I}, v >$ is a critical instance for the WAP and let G' be the graph that corresponds to the instance $< \mathcal{J}, \mathcal{I}, v - \epsilon >$, for any sufficiently small constant $\epsilon > 0$ in accordance with the Lemma 3. Then, any minimum (s,t)-cut $\mathcal{C}'$ of G' contains exactly:*

i. one arc of every path J_j, I_i, t for any critical job J_j,
ii. the arc (s, J_j) for each non-critical job J_j.

We are now ready to give a high level description of our algorithm. When the algorithm begins, we assume that the optimal schedule consumes a large amount of energy and all jobs are executed with the same speed s_{UB}. This speed value is selected so that there exists a feasible schedule executing all jobs with equal speed s_{UB}. Then, the algorithm decreases the common speed of all jobs up to a point where no further reduction is possible so as to obtain a feasible schedule. At this point, all jobs are assumed to be executed with the same speed, which is critical, and there is at least one job that cannot be executed with speed less than this, in any feasible schedule. The jobs that cannot be executed with speed less than the critical one form the current set of critical jobs. So, the critical job(s) is (are) assigned the critical speed and is (are) ignored thereafter. In what follows, the algorithm considers the subproblem in which some jobs are omitted (critical jobs), because they are already assigned the lowest speed possible (critical speed) so that they can be feasibly executed, and there are less than m processors during some intervals because these processors are dedicated to the omitted jobs.

In more detail, the algorithm schedules the jobs in a sequence of $\ell \leq n$ discrete steps. In the initial step, it identifies the set $\mathcal{J}_{crit}$ of critical jobs which are the ones executed with the maximum speed in the optimal schedule. To do so, it first computes the speed s_{crit} that these jobs are assigned, i.e. the

speed for which $< \mathcal{J}, \mathcal{I}, s_{crit} >$ is a critical WAP instance. The value s_{crit} is determined by performing binary search in the interval $[s_{LB}, s_{UB}]$, where s_{LB} is a speed value such $< \mathcal{J}, \mathcal{I}, s_{LB} >$ is infeasible. Once s_{crit} is determined, the critical jobs are identified by computing a maximum flow on the infeasible WAP instance $< \mathcal{J}, \mathcal{I}, s_{crit} - \epsilon >$. The value ϵ is selected so that it satisfies the requirement of the Lemma 3, i.e. the feasible, critical WAP instance $< \mathcal{J}, \mathcal{I}, s_{crit} >$ and the infeasible WAP instance $< \mathcal{J}, \mathcal{I}, s_{crit} - \epsilon >$ have the same set of critical jobs. Based on the maximum flow computation, the algorithm determines the critical jobs using Lemma 4 which indicates that there is a unique minimum cut on the corresponding graph of $< \mathcal{J}, \mathcal{I}, s_{crit} - \epsilon >$, for any $\epsilon > 0$ upper bounded as indicated by Lemma 3. Note that, if we choose $\epsilon = 0$, then the graph of $< \mathcal{J}, \mathcal{I}, s_{crit} >$ has multiple different minimum cuts with respect to the number of (s, J_j) edges, because it is a feasible WAP instance, and identifying the critical jobs is not obvious. This observation is the reason for detecting the critical jobs on the graph of $< \mathcal{J}, \mathcal{I}, s_{crit} - \epsilon >$. Once the algorithm identifies the critical jobs, it goes on with the remaining jobs and machine-intervals in the same manner. Algorithm 1 is a pseudocode of the proposed algorithm.

Next, we derive a valid, concrete value ϵ. Given a value ϵ in accordance with Lemma 3, Algorithm 1 produces the distinct job speed values $s_1 > s_2 > \ldots > s_\ell$. Because these speed values are critical speeds of critical jobs in critical WAP instances, they must belong to a well-defined discrete, exponential set as indicated by the following lemma.

Lemma 5 *Let $s_{j'}$ be the speed that Algorithm 1 assigns to job $J_{j'} \in \mathcal{J}$. It holds that $s_{j'} = (\sum_{J_j \in \mathcal{S}} w_j)/(\sum_{I_i \in \mathcal{I}} x_i |I_i|)$, for some subset $\mathcal{S} \subseteq \mathcal{J}$ of jobs and numbers $x_i \in [0, m]$ of machines $\forall I_i \in \mathcal{I}$.*

Proof When job $J_{j'}$ is assigned a speed by the algorithm, it is a critical job for some critical WAP instance $< \mathcal{J}', \mathcal{I}', v' >$ and $s_{j'} = v'$. Let $\mathcal{S}$ be the set of critical jobs in $< \mathcal{J}', \mathcal{I}', v' >$ and $\sum_{I_i \in \mathcal{I}} x_i |I_i|$ the sum of the arcs of the (J_j, I_i, t) paths in the Lemma 4 minimum cut. By the definition of criticality, it must be the case that $s_{j'} = (\sum_{J_j \in \mathcal{S}} w_j)/(\sum_{I_i \in \mathcal{I}} x_i |I_i|)$.

In what follows, we derive a suitable value for ϵ assuming that input data are integers. However, the arguments can be adapted to obtain an alternative, valid ϵ in the case of bounded rational input data. Lemma 3 selects a value ϵ such that the feasible, critical WAP instance $< \mathcal{J}, \mathcal{I}, v >$ and the infeasible instance $< \mathcal{J}, \mathcal{I}, v - \epsilon >$ have the same set of critical jobs, and thus the same set of non-critical jobs. In each step of the algorithm, this requirement for the value ϵ is equivalent to $s_k - s_{k+1} \geq \epsilon$. Otherwise, by the way the algorithm computes s_k and s_{k+1} and the way ϵ is obtained in the proof of Lemma 3, the corresponding WAP instances $< \mathcal{J}, \mathcal{I}, v >$ and $< \mathcal{J}, \mathcal{I}, v - \epsilon >$ in the computation of s_k would not have the same set of non-critical jobs, i.e. we would have a contradiction. Because the input data are integral and the fact that the speed values produced by the algorithm belong to a discrete set as indicated by Lemma 5, Lemma 6 derives an appropriate ϵ value.

Lemma 6 *Consider a pair of jobs $J_{j'}, J_{j''} \in \mathcal{J}$ for which Algorithm 1 assigns speeds $s_{j'}$ and $s_{j''}$ such that $s_{j'} \neq s_{j''}$. Without loss of generality, suppose that $s_{j'} > s_{j''}$. Assuming that the input data are integral, if $\epsilon = 1/2m^2L^2$, where $L = d_{\max} - r_{\min}$, then it holds that $s_{j'} - s_{j''} > \epsilon$.*

Proof By Lemma 5, $s_{j'} = (\sum_{J_j \in S'} w_j)/(\sum_{I_i \in \mathcal{I}} x'_i|I_i|)$ and $s_{j''} = (\sum_{J_j \in S''} w_j)/(\sum_{I_i \in \mathcal{I}} x''_i|I_i|)$, where $\mathcal{S}', \mathcal{S}'' \subseteq \mathcal{J}$ and $x'_i, x''_i \in [0, m]$ for $I_i \in \mathcal{I}$. Then, it holds that

$$\begin{aligned} s_{j'} - s_{j''} &= \frac{\sum_{J_j \in S'} w_j}{\sum_{I_i \in \mathcal{I}} x'_i|I_i|} - \frac{\sum_{J_j \in S''} w_j}{\sum_{I_i \in \mathcal{I}} x''_i|I_i|} \\ &= \frac{\left(\sum_{J_j \in S'} w_j\right)\left(\sum_{I_i \in \mathcal{I}} x''_i|I_i|\right) - \left(\sum_{J_j \in S''} w_j\right)\left(\sum_{I_i \in \mathcal{I}} x'_i|I_i|\right)}{\left(\sum_{I_i \in \mathcal{I}} x'_i|I_i|\right)\left(\sum_{I_i \in \mathcal{I}} x''_i|I_i|\right)} \\ &> \frac{1/2}{\left(\sum_{I_i \in \mathcal{I}} x'_i|I_i|\right)\left(\sum_{I_i \in \mathcal{I}} x''_i|I_i|\right)} > \frac{1}{2m^2L^2}, \end{aligned}$$

where the first inequality holds because the input data are integers and $s'_j \neq s_{j''}$.

Note that the binary search procedure may compute s_{crit} subject to ϵ accurracy. However, once the set $\mathcal{J}_{crit}$ of critical jobs has been determined, we may compute the critical speed exactly as follows. For each interval I_i, let m_i be the number of available processors. Then, $\min\{|\mathcal{J}_{crit}|, m_i\}$ processors are occupied by the jobs $\mathcal{J}_{crit}$ which are scheduled with constant speed so as to occupy these processors in every interval. This last speed is equal to

$$s_{crit} = \frac{\sum_{J_j \in \mathcal{J}_{crit}} w_j}{\sum_{I_i \in \mathcal{I}} \min\{|\mathcal{J}_{crit}|, m_i\}|I_i|} \tag{13}$$

Due to the convexity of the speed-to-power function, we know that each job J_j cannot be executed with speed less than its density $\delta_j = \frac{w_j}{d_j - r_j}$ in any optimal schedule. Therefore, given a set of jobs $\mathcal{J}$, we know that there does not exist an optimal schedule that executes all jobs with a speed $s < \max_{J_j \in \mathcal{J}}\{\delta_j\}$. Also, observe that if all jobs have speed $s = \max_{I_i \in \mathcal{I}}\{\frac{1}{|I_i|}\sum_{J_j \in \mathcal{J}} w_j\}$, then we can construct a feasible schedule. These bounds define the search space of the binary search performed in the initial step. In the next step the critical speed of the previous step is an upper bound on the speed of all remaining jobs and a lower bound is the maximum density among them. We use these updated bounds to perform the binary search of the current step and we go on like that.

Theorem 2 *Given a sufficiently small constant ϵ satisfying Lemma 4, Algorithm 1 produces an optimal schedule.*

Proof We will show that any schedule constructed by the algorithm satisfies the properties of Lemma 1. Let $G^{(k)}$ be the graph maintained by the algorithm at the beginning of the k-th step.

Algorithm 1

1: $s_{UB} = \max_i\{\frac{1}{|I_i|}\sum_{J_j \in \mathcal{A}_i} w_j\}$, $s_{LB} = \max_{J_j \in \mathcal{J}}\{\delta_j\}$
2: **while** $\mathcal{J} \neq \emptyset$ **do**
3: Find the maximum speed v so that the instance $< \mathcal{J}, \mathcal{I}, v >$ of the WAP is not feasible, using binary search in the interval $[s_{LB}, s_{UB}]$ with repeated maximum flow computations until the size of the interval is at most ϵ (this ϵ is small enough to satisfy Lemma 3).
4: Determine the set of critical jobs $\mathcal{J}_{crit}$ by computing a minimum (s,t)-cut in the graph G' that corresponds to the instance $< \mathcal{J}, \mathcal{I}, v - \epsilon >$ based on Lemma 4.
5: Assign to each job in J_{crit} the speed computed according to Equation (13).
6: $\mathcal{J} = \mathcal{J} \backslash \mathcal{J}_{crit}$.
7: Update the set of available processors for each interval $I_i \in \mathcal{I}$.
8: $s_{UB} = s_{crit}$, $s_{LB} = \max_{J_j \in \mathcal{J} \backslash \mathcal{J}_{crit}}\{\delta_j\}$
9: Apply an optimal algorithm for $P|pmtn, r_j, d_j|-$ to schedule the jobs, where each job J_j has processing time w_j/s_j.

We begin with the proof of property 1. Consider an interval I_i s.t. $|\mathcal{A}_i| \leq m$ and a job $J_j \in \mathcal{A}_i$. Assume that J_j becomes critical in the k-th step. Then, either the edge (J_j, I_i) or the edge (I_i, t) must belong to a minimum (s,t)-cut of $G^{(k)}$. Since $|\mathcal{A}_i| \leq m$, only the first is possible which means that $t_{i,j} = |I_i|$ in the algorithm's schedule.

In the remainder of the proof, we consider an interval I_i s.t. $|\mathcal{A}_i| > m$. Assume that a job $J_j \in \mathcal{A}_i$ is scheduled in the k-th iteration. Then, either the edge (J_j, I_i) or the edge (I_i, t) belongs to a minimum (s,t)-cut of $G^{(k)}$. Since there are more than m jobs active during I_i, we conclude that $\sum_{J_j \in \mathcal{A}_i} t_{i,j} = m|I_i|$ in the algorithm's schedule and property 2i is true.

For property 2ii, consider two jobs J_j and $J_{j'}$, active during I_i, such that $0 < t_{i,j} < |I_i|$ and $0 < t_{i,j'} < |I_i|$. We will show that the jobs are assigned equal speeds by the algorithm. For this, it suffices to show that they are assigned a speed at the same step. So, assume for contradiction that J_j becomes critical before $J_{j'}$, at the k-th step. Then, either the arc (J_j, I_i) or the arc (I_i, t) belongs to a minimum (s,t)-cut $\mathcal{C}$ in $G^{(k)}$. Since $0 < t_{i,j} < |I_i|$, we know that there exists a maximum (s,t)-flow in $G^{(k)}$ such that $0 < f_{(J_j, I_i)} < |I_i|$. Thus, it is the arc (I_i, t) that belongs in $\mathcal{C}$. Consequently, in $G^{(k)}$, the edge (I_i, t) is saturated by any maximum (s,t)-flow, and as a result, all the processors during the interval I_i are dedicated to the execution of some tasks at the end of the k-th step. Hence, $J_{j'}$ cannot be scheduled in a subsequent step.

For property 2iii, consider the case where $t_{i,j} = 0$ for a job $J_j \in \mathcal{A}_i$ and assume that J_j becomes critical at the k-th step. Then, either (I_i, t) does not appear in $G^{(k)}$ or (I_i, t) belongs to a minimum (s,t)-cut of $G^{(k)}$. If none of these was true, then J_j would not be critical in $G^{(k)}$. Therefore, J_j cannot have speed greater than any job scheduled during I_i.

Next, let J_j be a job with $t_{i,j} = |I_i|$ and assume that it is assigned a speed at the k-th step. Given our previous observations, this cannot happen after a step where a job $J_{j'}$ with $0 < t_{i,j'} < |I_i|$ or $t_{i,j'} = 0$ is scheduled.

We turn, now, our attention to the running time of the algorithm. Because of Lemma 2, at least one job is scheduled at each step of the algorithm. Therefore, there will be at most n steps. Assume that U is an upper bound on the speed of any job in the optimal schedule, e.g. $U = \max_{I_i \in \mathcal{I}}\{\frac{1}{|I_i|}\sum_{J_j \in \mathcal{J}} w_j\}$. Then, the binary search needs to check $O(\log \frac{U}{\epsilon})$ values of speed to determine the next critical speed at one step, where ϵ is small enough so that Lemmas 3 and 4 are satisfied. That is, BAL performs $O(\log \frac{U}{\epsilon})$ maximum flow calculations at each step. Thus, the overall complexity of our algorithm is $O(nf(n)\log \frac{U}{\epsilon})$, where $f(n)$ is the complexity of computing a maximum flow in a graph with $O(n)$ vertices.

In a graph with V vertices and E edges, a maximum flow can be computed in $O(V^2E)$ time using (i) the push-relabel algorithm [1], or (ii) Dinic's blocking flow algorithm [14]. Using the dynamic tree data structure by Sleator and Tarjan [21], a maximum flow is computed in $O(VE\log V)$ time. The network of Algorithm 1 contains $V = O(n)$ vertices and $E = O(n^2)$ edges. Based on this result, we conclude the following corollary.

Corollary 1 *There exists an optimal algorithm for the speed scaling problem on parallel processors with migration, running in* $O(n^4 \log(nm^2L^2U))$ *time.*

6 Conclusion

We studied the multiprocessor speed scaling problem of minimizing the energy with migrations. We proposed a combinatorial polynomial-time algorithm based on a reduction to the maximum flow problem. Since there is not much work on problems with migrations there are many directions and speed problems to be considered in a multicriteria context. All these problems seem to be very interesting and might require new algorithmic techniques because of their continuous nature. We believe that the approach used in our paper might be useful in this direction.

Acknowledgements We thank Alexander Kononov for helpful discussions on this work. We also thank the anonymous referees for their very useful comments on improving the algorithm's presentation and the analysis of the optimality conditions.

References

1. R.K. Ahuja, T.L. Magnanti, J.B. Orlin. Network flows: theory, algorithms, and applications. Prentice-Hall, 1993.
2. S. Albers. Energy Efficient Algorithms. Communications of the ACM 53(5), p. 86-96 2010.
3. S. Albers. Algorithms for Dynamic Speed Scaling. International Symposium of Theoretical Aspects of Computer Science (*STACS'11*), LIPIcs 9, p. 1-11, 2011.
4. S. Albers, A. Antoniadis, G. Greiner. On Multi-Processor Speed Scaling with Migration. ACM Symposium on Parallelism in Algorithms and Architectures (*SPAA'11*), p. 279-288, 2011.

5. S. Albers and H. Fujiwara. Energy Efficient Algorithms for Flow Time Minimization. ACM Transactions on Algorithms 3(4):49, 2007.
6. S. Albers, F. Muller and S. Schmelzer. Speed Scaling on Parallel Processors. ACM Symposium on Parallelism in Algorithms and Architectures (*SPAA'07*), p. 289-298, 2007.
7. E. Bampis, A. Kononov, D. Letsios, G. Lucarelli and M. Sviridenko. Energy Efficient Scheduling and Routing via Randomized Rounding. International Conference on Foundations of Software Technology and Theoretical Computer Science (*FSTTCS'13*), p. 449-460, 2013.
8. N. Bansal, T. Kimbrel and K. Pruhs. Speed Scaling to Manage Energy and Temperature. Journal of the ACM 54(1):3, 2007.
9. E. Bampis, D. Letsios and G. Lucarelli. Green Scheduling, Flows and Matchings. International Symposium on Algorithms and Computation (*ISAAC*'12), p. 106-115, 2012.
10. Ph. Baptiste, E. Néron, F. Sourd. Modèles et Algorithmes en Ordonnancement, Ellipses, 2004.
11. B. Bingham and M. Greenstreet. Energy Optimal Scheduling on Multiprocessors with Migration. IEEE International Symposium on Parallel and Distributed Processing with Applications (*ISPA'08*), p. 153-161, 2008.
12. H. L. Chan, J. W. T. Chan, T. W. Lam, L. K. Lee, K. S. Mak and P. W. H. Wong. Optimizing Throughput and Energy in Online Deadline Scheduling. ACM Transactions on Algorithms 6(1):10, 2009.
13. J.J. Chen, H. R. Hsu, K. H. Chuang, C. L. Yang, A. C. Pang and T. W. Kuo. Multiprocessor Energy Efficient Scheduling with Task Migration Considerations. Euromicro Conference on Real-Time Systems (*ECRTS'04*), p. 101-108, 2004.
14. Yefim Dinitz. Dinitz' Algorithm: The Original Version and Even's Version. In Theoretical Computer Science, Essays in Memory of Shimon Even, p. 218-240, 2006.
15. G. Greiner, T. Nonner and A. Souza. The Bell is Ringing in Speed Scaled Multiprocessor Scheduling. ACM Symposium on Parallelism in Algorithms and Architectures (*SPAA'09*), p. 11-18, 2009.
16. S. Irani and K. Pruhs. Algorithmic Problems in Power Management. ACM Sigact News 36(2), p. 63-76, 2005.
17. J. Kleinberg, E. Tardos, Algorithm Design, Addison Wesley, 2006.
18. Robert McNaughton. Scheduling with deadlines and loss functions. Management Science 6(1), p. 1-12, 1959.
19. A. Nemirovski, I. Nesterov and Y. Nesterov. Interior Point Polynomial Algorithms in Convex Programming. Society for Industrial and Applied Mathematics, 1994.
20. K. Pruhs, P. Uthaisombut and G. Woeginger. Getting the Best Response for your Erg. ACM Transaction on Algorithms 4(3):38, 2008.
21. D.D. Sleator and R.E. Tarjan. A data structure for dynamic trees. Journal of Computer and System Sciences, 26(3):362-391, 1983.
22. F. Yao, A. Demers and S. Shenker. A Scheduling Model for Reduced CPU Energy. IEEE Symposium on Foundations of Computer Science (*FOCS'95*), p. 374-382, 1995.

Appendix: Convex Programming and KKT Conditions

A optimization problem in the following form is called a *convex optimization problem* if all the functions $f, g_1, \ldots, g_m : \mathbb{R}^n \to \mathbb{R}$ are convex.

$$\begin{aligned} &\min f(x) \\ &g_i(x) \leq 0 \qquad 1 \leq i \leq m \\ &x \in \mathbb{R}^n \end{aligned}$$

In what follows, we state the KKT (Karush, Kuhn, Tucker) conditions which are necessary and sufficient for optimality in convex programming (see

[19]). Consider any convex program in the above form where all functions all functions g_i are differentiable in their domain. Suppose that the program is strictly feasible, i.e. there is a point x such that $g_i(x) < 0$, for all $1 \leq i \leq m$. To each constraint $g_i(x) \leq 0$, we associate a dual variable λ_i. Then, the KKT conditions are expressed as follows

$$g_i(x) \leq 0 \qquad 1 \leq i \leq m \tag{11}$$

$$\lambda_i \geq 0 \qquad 1 \leq i \leq m \tag{12}$$

$$\lambda_i \cdot g_i(x) = 0 \qquad 1 \leq i \leq m \tag{13}$$

$$\nabla f(x) + \sum_{i=1}^{m} \lambda_i \cdot \nabla g_i(x) = 0 \tag{14}$$

KKT conditions are necessary and sufficient for the solutions $x \in \mathbf{R}^n$ and $\lambda \in \mathbf{R}^m$ to be primal and dual optimal, where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)$. The conditions (11) are known as primal feasible, (12) as dual feasible, (13) as complementary slackness and (14) as stationarity conditions, respectively.