Energy Minimization via a Primal-Dual Algorithm for a Convex Program

Evripidis Bampis[1,⋆,⋆⋆], Vincent Chau[2,⋆], Dimitrios Letsios[1,2,⋆], Giorgio Lucarelli[1,2,⋆,⋆⋆], and Ioannis Milis[3,⋆⋆]

[1] LIP6, Université Pierre et Marie Curie, France.
{Evripidis.Bampis, Giorgio.Lucarelli}@lip6.fr
[2] IBISC, Université d'Évry, France.
{vincent.chau,dimitris.letsios}@ibisc.univ-evry.fr
[3] Dept. of Informatics, AUEB, Athens, Greece.
milis@aueb.gr

Abstract. We present an optimal primal-dual algorithm for the energy minimization preemptive open-shop problem in the speed-scaling setting. Our algorithm uses the approach of Devanur et al. [JACM 2008], by applying the primal-dual method in the setting of convex programs and KKT conditions. We prove that our algorithm converges and that it returns an optimal solution, but we were unable to prove that it converges in polynomial time. For this reason, we conducted a series of experiments showing that the number of iterations of our algorithm increases linearly with the number of jobs, n, when n is greater than the number of machines, m. We also compared the speed of our method with respect to the time spent by a commercial solver to directly solve the corresponding convex program. The computational results give evidence that for $n > m$, our algorithm is clearly faster. However, for the special family of instances where $n = m$, our method is slower.

1 Introduction

The *primal-dual* method has been extensively used for obtaining optimal [7, 8, 13] and approximate [9, 14] solutions for many well known optimization problems. It has been mainly applied for problems formulated as linear programs. Only recently Devanur et al. [6] applied the primal-dual paradigm in the more general setting of convex programming and the Karush-Kuhn-Tucker (KKT) conditions. Our work is in the same vein. We explore the primal-dual paradigm in the context of energy minimization scheduling with respect to the *speed-scaling* model. In this model the speed of each processor can dynamically change at any time,

⋆ Partially supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010, and a PHC CAI YUANPEI France-China bilateral project.

⋆⋆ Partially supported by the project ALGONOW, co-financed by the European Union (European Social Fund - ESF) and Greek national funds, through the Operational Program "Education and Lifelong Learning", under the program THALES.

while the energy consumption is a convex function of the speed. More formally, if a processor runs at speed $s(t)$ at time t, then the power needed is $P(t) = s(t)^\alpha$, where $\alpha > 1$ is a machine-dependent constant.[4] The energy consumption of the processor is equal to the integral of the power over time, i.e., $E = \int P(t)dt$. Moreover, the processor's speed is the rate at which work is executed and, thus, the total of work amount executed by a processor is the integral of its speed, i.e. $w = \int s(t)dt$.

We focus on the speed-scaling preemptive open-shop problem for which there is a natural formulation as a convex program. In the energy minimization speed-scaling preemptive open-shop problem, we are given a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ and a set of m processors $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$. Each job consists of operations that have to run on different processors. Operations of the same job cannot be executed at the same time. The operation O_{ij} of the job J_j has to be executed on processor M_i and it has an amount of work $w_{ij} \geq 0$. Note that, it is not necessary for each job to have an operation to all processors; in this case $w_{ij} = 0$. The operations may be preempted, that is they can be interrupted and continue their execution later. The goal is to minimize the total energy consumed such that all operations are completed before a given deadline d. Extending the Graham's three-field notation [11] for scheduling problems, we denote our problem by $O|pmtn, d|E$.

Related Work. The preemptive open-shop problem has been extensively studied in the classical setting (without caring about the energy consumption). In this setting, each operation O_{ij} has a processing time p_{ij}, instead of a work. When the goal is the minimization of the length of the schedule (makespan) the problem $O|pmtn|C_{\max}$ can be solved in polynomial time [10]. Several other results for the preemptive open-shop can be found in [5]. The speed-scaling preemptive open-shop problem has beed studied, recently, by Bampis et al. [3], who presented a combinatorial algorithm for $O|pmtn, d|E$ based on a transformation to a convex cost flow problem.

In [12], Gupta et al. apply a primal-dual approach in order to obtain a constant factor competitive algorithm for a speed-scaling problem in the online setting. This algorithm is based on a primal-dual schema using the Lagrangian duality and it is quite different from our primal-dual approach.

There is a lot of work in the speed-scaling area, and the interested reader is referred to the recent reviews [1, 2].

Our contribution. In linear programming, the idea of the primal-dual approach is, in general, to modify the dual and the primal variables in turns, based on the complementary slackness conditions. In the convex programming setting, it is not possible to define a dual program as for the linear problems. However, there is always an optimal solution for any convex program in which the primal variables are related with the dual variables through some equality relationships known as the stationarity conditions. Therefore, by modifying the dual variables,

[4] In most applications, α is considered to be between two and three.

that correspond to Lagrangian multipliers, there is a direct impact on the values of the primal variables.

In Section 3 we formulate our problem as a convex program. Then, in Section 4, we propose a primal-dual algorithm and we prove that it is optimal and that it converges. Unfortunately, we are unable to prove that it converges in polynomial time. In Section 5, we present a series of experiments showing that the number of iterations of our algorithm increases linearly with the number of jobs when $n > m$. We are also interested in the comparison of the execution time of our method with respect to the time spent by a commercial solver to directly solve the corresponding convex program. The computational results show that for $n > m$, our algorithm is clearly faster. However, for the special family of instances where $n = m$, our method is slower.

2 Preliminaries

In most of the speed-scaling problems, due to the convexity of the speed-to-power function, each job/operation runs at a constant speed during its whole execution in an optimal schedule (see for example [15]). This observation holds also for our problem and its proof directly follows from the convexity of the power function. Note that, given the speed s_{ij} of the operation O_{ij}, the time needed for the execution of O_{ij} is $\frac{w_{ij}}{s_{ij}}$, while the energy consumed during its execution is $\frac{w_{ij}}{s_{ij}} s_{ij}^{\alpha} = w_{ij} s_{ij}^{\alpha-1}$.

In what follows in this paper, we will consider a relaxation Π' of our original problem Π, in which operations of the same job are allowed to be executed simultaneously but the sum of the execution times of the operations of the same job cannot exceed d. A solution of this problem gives the speeds of the operations, without determining their order. Clearly, for an optimal solution E' of Π', it holds that $E' \leq E$, where E is the energy consumption in an optimal solution for Π.

In the following sections we formulate the relaxed problem as a convex program and we propose a primal-dual algorithm to find an optimal solution for it. A solution for Π' determines the speeds of operations, and hence their processing times. Then, we can run a polynomial algorithm for $O|pmtn|C_{\max}$ (see for example [10]) to get a feasible open-shop solution for our problem Π. This procedure is described formally in Algorithm 1.

Algorithm 1

1: Solve optimally the relaxed problem Π' to define the speeds of operations;
2: For each O_{ij} set processing time $p_{ij} = \frac{w_{ij}}{s_{ij}}$;
3: Run the algorithm proposed in [10] for $O|pmtn|C_{\max}$;

Theorem 1. *Algorithm 1 returns an optimal schedule for $O|pmtn, d|E$.*

Proof. In an optimal solution for Π', all jobs and processors are active for time at most d, while it is easy to see that there is at least one job or processor with processing time exactly d. If not, we can decrease the speeds of all operations and get a feasible schedule for Π' of smaller energy consumption, which is a contradiction as we have considered an optimal schedule. Moreover, it is known (e.g. [5]) that any optimal solution for $O|pmtn|C_{\max}$ has length equal to $\max\{\max_i \sum_{O_{ij}\in M_i} p_{ij}, \max_j \sum_{O_{ij}\in J_j} p_{ij}\}$, which in our case is d. Thus, Algorithm 1 returns a feasible open-shop solution for Π, i.e., a solution where no operations of the same job are executed simultaneously and the completion time of all operations is at most d. In other words, the algorithm for $O|pmtn|C_{\max}$ produces a feasible solution of Π given the speeds obtained by a feasible solution for Π', and hence the energy consumed for Π is equal to E'. As the speeds, and hence the processing times, of operations are selected in Line 1 in such a way that E' is minimized, the theorem holds. □

3 Convex Programming Formulation and the KKT conditions

In this section we first formulate our relaxed problem as a convex program.

$$\begin{aligned}
\min \sum_{O_{ij}\in J_j} \sum_{O_{ij}\in M_i} & w_{ij} s_{ij}^{\alpha-1} \\
\sum_{O_{ij}\in M_i} \frac{w_{ij}}{s_{ij}} &\le d \qquad 1 \le i \le m \qquad (1)\\
\sum_{O_{ij}\in J_j} \frac{w_{ij}}{s_{ij}} &\le d \qquad 1 \le j \le n \qquad (2)\\
s_{ij} &\ge 0 \qquad O_{ij} \in J_j, O_{ij} \in M_i
\end{aligned}$$

Constraints (1) and (2) do not allow a job and a processor, respectively, to be active for a time period greater than d.

The Karush-Kuhn-Tucker conditions are necessary and sufficient conditions [4] for a feasible solution of our convex program to be optimal. Note that, the β_i's, $1 \le i \le m$, correspond to the Lagrangian multipliers for the Constraints (1) and the γ_j's, $1 \le j \le n$, correspond to the Lagrangian multipliers for the Constraints (2).

Stationarity conditions:

$$\begin{aligned}
&\nabla\left(\sum_{O_{ij}\in J_j}\sum_{O_{ij}\in M_i} w_{ij} s_{ij}^{a-1}\right) + \sum_{i=1}^{m} \beta_i \cdot \nabla\left(\sum_{O_{ij}\in M_i} \frac{w_{ij}}{s_{ij}} - d\right)\\
&+\sum_{j=1}^{n} \gamma_j \cdot \nabla\left(\sum_{O_{ij}\in J_j} \frac{w_{ij}}{s_{ij}} - d\right) + \sum_{O_{ij}\in J_j}\sum_{O_{ij}\in M_i} \delta_{ij} \cdot \nabla(-s_{ij}) = 0
\end{aligned}$$

or equivalently

$$\sum_{O_{ij}\in J_j}\sum_{O_{ij}\in M_i}\left(w_{ij}(a-1)s_{ij}^{a-2}-(\beta_i+\gamma_j)\frac{w_{ij}}{s_{ij}^2}-\delta_{ij}\right)\nabla s_{ij}=0 \tag{3}$$

Complementary slackness conditions:

$$\beta_i\cdot\left(\sum_{O_{ij}\in M_i}\frac{w_{ij}}{s_{ij}}-d\right)=0 \qquad 1\le i\le m \tag{4}$$

$$\gamma_j\cdot\left(\sum_{O_{ij}\in J_j}\frac{w_{ij}}{s_{ij}}-d\right)=0 \qquad 1\le j\le n \tag{5}$$

$$\delta_{ij}\cdot(-s_{ij})=0 \qquad O_{ij}\in J_j, O_{ij}\in M_i \tag{6}$$

Note that operations with work $w_{ij}=0$ are not counted into the above sums. Then, as each operation has a work $w_{ij}>0$ to execute, it holds that $s_{ij}>0$, and hence, by condition (6) we have that $\delta_{ij}=0$. Thus, Equation (3) can be reformulated as

$$s_{ij}^{\alpha}=\frac{\beta_i+\gamma_j}{\alpha-1} \qquad O_{ij}\in J_j, O_{ij}\in M_i \tag{7}$$

As we already mentioned in the introduction, the KKT conditions, and especially the stationarity conditions, give a relation between the primal and the dual variables. Indeed, Equations (7) directly connect our primal variables s_{ij} with our dual variables β_i and γ_j. Intuitively, each dual variable β_i, $1\le i\le m$, can be considered as the contribution of the processor M_i to the speed of the operations O_{ij}, $1\le j\le n$. In a similar way, each dual variable γ_j, $1\le j\le n$, can be considered to be the contribution of the job J_j to the speed of the operations O_{ij}, $1\le i\le m$.

4 The Primal-Dual Algorithm

In this section we present a combinatorial algorithm based on the primal-dual approach. The main idea of the algorithm is to determine the values of dual variables, β_i and γ_j, and hence the speeds of operations, through a primal-dual scheme. Our algorithm initializes the dual variables according to the following proposition that provides upper and lower bounds for them.

Proposition 1.
(i) For each β_i, $1\le i\le m$, it holds that $0\le\beta_i\le(\alpha-1)\left(\frac{\sum_{O_{ij}\in M_i}w_{ij}}{d}\right)^{\alpha}$.
(ii) For each γ_j, $1\le j\le n$, it holds that $0\le\gamma_j\le(\alpha-1)\left(\frac{\sum_{O_{ij}\in J_j}w_{ij}}{d}\right)^{\alpha}$.

Proof. The lower bounds follow by the definition of β_i's and γ_j's.

For the upper bound on β_i's, consider a processor M_i, $1 \leq i \leq m$. As we search for an upper bound we can assume that $\beta_i > 0$. Hence, by the complementary slackness conditions (4) and applying the stationarity conditions (7), in an optimal solution it holds that

$$\sum_{O_{ij} \in M_i} \frac{w_{ij}}{s_{ij}} - d = 0 \Leftrightarrow \sum_{O_{ij} \in M_i} \frac{w_{ij}}{\sqrt[\alpha]{\frac{\beta_i + \gamma_j}{\alpha - 1}}} = d$$

To obtain the upper bound on β_i, we can consider that the speeds of all operations $O_{ij} \in M_i$ depend only on the contribution of the processor M_i, that is $\gamma_j = 0$ for all $O_{ij} \in M_i$. Hence, we have that

$$\sum_{O_{ij} \in M_i} \frac{w_{ij}}{\sqrt[\alpha]{\frac{\beta_i}{\alpha - 1}}} \geq d \Leftrightarrow \beta_i \leq (\alpha - 1) \left(\frac{\sum_{O_{ij} \in M_i} w_{ij}}{d} \right)^{\alpha}$$

The same arguments hold for the upper bounds on γ_j's. □

Based on the previous proposition, we initialize each dual variable β_i, $1 \leq i \leq m$, to its lower bound and each dual variable γ_j, $1 \leq j \leq n$, to its upper bound. Given these initial values, the obtained schedule may not be feasible. More specifically, the processing time of some processors may be more than d, i.e., $\sum_{O_{ij} \in M_i} \frac{w_{ij}}{\sqrt[\alpha]{\frac{\gamma_j}{\alpha - 1}}} > d$. For such a processor M_i, we increase β_i such that the processing time of M_i is exactly d, i.e., $\sum_{O_{ij} \in M_i} \frac{w_{ij}}{\sqrt[\alpha]{\frac{\beta_i + \gamma_j}{\alpha - 1}}} = d$. We refer to this step as an "infeasible-to-feasible" step. The increasing of β_i's has as a result some jobs to become non-tight, i.e., $\sum_{O_{ij} \in J_j} \frac{w_{ij}}{\sqrt[\alpha]{\frac{\beta_i + \gamma_j}{\alpha - 1}}} < d$. For such a job J_j, we decrease γ_j such that to be equal to the maximum between zero (respecting our definition) and the value of γ_j needed so that J_j becomes tight again, i.e., $\sum_{O_{ij} \in J_j} \frac{w_{ij}}{\sqrt[\alpha]{\frac{\beta_i + \gamma_j}{\alpha - 1}}} = d$. We refer to this step as a "non-tight-to-tight" step. Thus, the decreasing of γ_j's has as a result some processors to become non-feasible, and so on. The criterion to terminate this procedure is when after a "non-tight-to-tight" step all the complementary slackness conditions are satisfied. A formal description of the above procedure is given in Algorithm 2.

Note that, the algorithm modifies a dual variable β_i only if the processor M_i is non-feasible in such a way to make it feasible (and tight). To do this, the speed of each operation $O_{ij} \in M_i$ is increased through the increasing of β_i. By the definition of the algorithm, M_i can be in a feasible and non-tight state only if $\beta_i = 0$. In a similar way the algorithm modifies a dual variable γ_j only if the job J_j is non-tight (and feasible) in such a way to make it tight. To do this, the speed of each operation $O_{ij} \in J_j$ is decreased through the decreasing of γ_j. By the definition of the algorithm, J_j cannot be in an infeasible state. Based on these observations, the following proposition follows.

Algorithm 2

1: For each i, $1 \leq i \leq m$, set $\beta_i = 0$;
2: For each j, $1 \leq j \leq n$, set $\gamma_j = (\alpha - 1)\left(\frac{\sum_{O_{ij} \in J_j} w_{ij}}{d}\right)^{\alpha}$;
3: **while** the complementary slackness conditions are not satisfied **do**
4: **for** each i, $1 \leq i \leq m$, such that the processor M_i is not feasible **do**
5: Choose β_i such that $\left(\sum_{O_{ij} \in J_j} \frac{w_{ij}}{\sqrt[\alpha]{\frac{\beta_i + \gamma_j}{\alpha - 1}}} - d\right) = 0$;
6: **for** each j, $1 \leq j \leq j$, such that the job J_j is not tight **do**
7: Choose the maximum value of γ_j such that $\gamma_j \cdot \left(d - \sum_{O_{ij} \in J_j} \frac{w_{ij}}{\sqrt[\alpha]{\frac{\beta_i + \gamma_j}{\alpha - 1}}}\right) = 0$;

Proposition 2.
(i) For each i, $1 \leq i \leq m$, the value of β_i is always non-decreasing.
(ii) For each j, $1 \leq j \leq n$, the value of γ_j is always non-increasing.

Theorem 2. *Algorithm 2 converges to an optimal solution of the relaxed problem.*

Proof. In each iteration the algorithm modifies at least one dual variable; otherwise the complementary slackness conditions are satisfied and the algorithm terminates. By Proposition 2 the modification of the dual variables is monotone, while by Proposition 1 there are well-defined lower and upper bounds for them. Therefore, the algorithm terminates.

In order to show that the algorithm converges in an optimal solution, we just have to show that the solution obtained satisfies the KKT conditions. The stationarity conditions (7) hold as for any operation O_{ij} the assigned speed by construction can be written as $s_{ij} = \sqrt[\alpha]{\frac{\beta_i + \gamma_j}{\alpha - 1}}$. The complementary slackness conditions (4) hold since after the final "non-tight-to-tight" step any processor M_i is either tight or its $\beta_i = 0$; if not then the algorithm would have executed a new iteration. The complementary slackness conditions (5) hold since in Line 7 we force $\gamma_j \cdot \left(d - \sum_{O_{ij} \in J_j} \frac{w_{ij}}{\sqrt[\alpha]{\frac{\beta_i + \gamma_j}{\alpha - 1}}}\right) = 0$. The complementary slackness conditions (6) hold since for any operation O_{ij} we have set $\delta_{ij} = 0$. □

5 Experimental Results

In this section we experimentally test our primal-dual algorithm towards two directions. The first direction is to observe the behavior of our algorithm when the size of the instance increases. The second direction is to compare the execution time of the primal-dual approach against the execution time of a baseline algorithm, that is a commercial solver that solves directly the corresponding convex program.

5.1 System Specification and Benchmark Generation

Our simulations have been performed on a machine with a CPU Intel Xeon X5650 with 8 cores, running at 2.67GHz. The operating system of the machine is a Linux Debian 6.0. We used Matlab with cvx toolbox. The solver used for the convex program is SeDuMi. For both our algorithm and the convex program, we set $\varepsilon = 10^{-7}$ to be the desired accuracy of the returned solution.

The instance of the problem consists of a matrix $m \times n$ that corresponds to the work of the operations, the value of α and the deadline d. However, we experiment with two more parameters: (i) the density p of the instance, that is the number of non-zero work operations, and (ii) the range $[1, w_{\max}]$ of the values of works.

We have considered several combinations for the parameters m, $1 \leq m \leq 50$, and n, $1 \leq n \leq 200$. For each combination, we have first decided randomly with probability p if there is a non-zero work operation in each position of the $m \times n$ matrix. The value of p has been selected to be 0.5 or 0.75 or 1. If the created instance did not correspond to the selected values of m and n, we rejected it and we replaced it by another. In other words, we reject a matrix iff there exists a line or a column in which each value is equal to zero. Then, for each operation with non-zero work, we selected at random an integer in the range of $[1, w_{\max}]$. Note here that $w_{\max}$ and the deadline d are strongly related. Indeed, given a matrix of works and a deadline d, if we increase all works and the deadline by the same factor, then the optimal solutions of the two instances will tend to have very similar (if not the same) speeds and energy consumption. For this reason, we have fixed the value of $d = 1000$ and we examined three different values for $w_{\max}$, i.e., $w_{\max} = 10$, $w_{\max} = 50$ and $w_{\max} = 100$. These values are selected, in general, in the direction of creating instances in which the average speed in the optimal solution is greater than one, almost equal to one and smaller than one, respectively. Finally, as in most applications the value of α is between two and three, we used three different values for it, that is $\alpha = 2$, $\alpha = 2.5$ and $\alpha = 3$.

For each combination of parameters we have repeated the experiments with 30 different matrices. All results we present below, concern the average of these 30 instances.

The benchmark as well as the code we used in our experiments are freely available at `http://todo.lamsade.dauphine.fr/spip.php?article85`.

5.2 Results

The main goal of our experiments is to study the behavior of the primal-dual algorithm when the size of the instance increases. However, during our experiments we noticed that the speed of convergence strongly depends on the relation between the number of jobs n and the number of processors m.

In Table 1, we show how the size of the instance affects the number of modifications of the dual variables made by the primal-dual algorithm. We observe that, if $n > m$ then the number of modifications increases linearly with the size

of the instance (see also Fig. 1). Moreover, the parameters α, $w_{\max}$ and p do not play any role to the number of modifications.

Note also that if $n < m$ then the number of modifications increases linearly with the size of the instance. In fact the two cases $n > m$ and $n < m$ should be symmetric. However, the initialization step of our algorithm breaks this symmetry. Recall that the algorithm initializes the dual variables that correspond to processors (β_i's) to zero and the dual variables that correspond to jobs (γ_j's) to their upper bounds. In the case where $n < m$, we expect to have all jobs tight and most of the processors non-tight in the optimal schedule of a random instance. Hence, the number of non-zero β_i's is expected to be very small. The initialization step helps in this direction, and this is the reason why the number of modifications is very small if $n < m$.

n	$m = 5$	$m = 10$	$m = 15$	$m = 20$	$m = 25$	$m = 30$	$m = 40$	$m = 50$
5	40101	1	2	2	2	2	2	2
10	151	279611	3	4	3	4	4	4
20	255	295	384	–	34	7	7	10
30	355	410	443	500	593	–	12	15
40	455	510	565	572	640	756	–	32
50	555	610	665	720	768	755	947	–
60	655	710	765	820	872	864	1040	1294
70	755	810	865	920	975	1030	1034	1250
100	1055	1110	1165	1220	1275	1330	1440	1495
150	1555	1610	1665	1720	1775	1830	1940	2050
200	2055	2110	2165	2220	2275	2330	2440	2550

Table 1. The number of modifications of the dual variables done by the primal-dual algorithm. The values of the table correspond to $\alpha = 2$, $w_{\max} = 10$, $p = 1$. Each entry of the table is the average over 30 instances. The empty entries correspond to cases with $m = n$ and take time longer than 30 minutes each and are interrupted.

However, if $n = m$ the behavior of our algorithm completely changes. For example, for $m = 10$ and $n = 10$ we need 279611 modifications, while for $m = 10$ and $n = 20$ we need only 295. Even more, for $m = n = 20$ the primal-dual algorithm does not even converge in 30 minutes. Furthermore, if $m = n$ then the parameters α, $w_{\max}$ and p affect the convergence of the algorithm. For example, in the case where $m = 10$ and $n = 10$, then the following table shows the number of modifications of the dual variables performed by our algorithm when we fix the two of the three parameters. Note that in the last line of the table, the algorithm did not terminate within the time threshold.

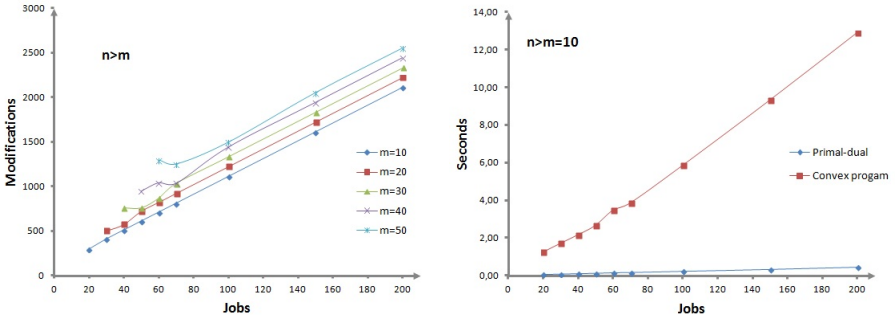
Parameters		Modifications
$\alpha = 2$, $w_{\max} = 10$	$p = 0.5$	344
	$p = 0.75$	23915
	$p = 1$	179611
$w_{\max} = 10$, $p = 1$	$\alpha = 2$	279611
	$\alpha = 2.5$	59785
	$\alpha = 3$	10716
$\alpha = 2$, $p = 1$	$w_{\max} = 10$	279611
	$w_{\max} = 50$	406608
	$w_{\max} = 100$	–

In Table 2 we give a comparison of the execution time of the primal-dual algorithm with the execution time of solving directly the convex program using the SeDuMi solver in Matlab. We observe again the difference between $n \neq m$ and $n = m$. In the first case, our algorithm highly outperforms the solver (see Fig. 1). In the second case, our algorithm does not even terminate within 30 minutes if $n = m = 20$, while the execution time of the solver is not affected. Note also that the solver's execution time as well as the execution time of the primal-dual algorithm when $n = m$ depend on the parameters α, $w_{\max}$ and p.

n	$m = 10$		$m = 20$		$m = 30$		$m = 40$		$m = 50$	
	CP	PD	CP	PD	CP	PD	CP	PD	CP	PD
5	0.59	0.00	0.99	0.00	1.41	0.01	1.83	0.01	2.11	0.01
10	1.22	147.93	1.26	0.01	1.81	0.01	2.42	0.01	2.59	0.01
20	1.25	0.06	3.12	–	2.57	0.02	3.11	0.02	3.92	0.03
30	1.72	0.08	2.58	0.12	5.57	–	4.36	0.03	5.30	0.04
40	2.17	0.10	3.28	0.13	4.38	0.21	8.31	–	6.48	0.05
50	2.67	0.12	4.00	0.16	5.19	0.19	6.72	0.33	11.49	–
60	3.47	0.15	4.96	0.18	6.72	0.23	8.39	0.29	9.87	0.47
70	3.86	0.16	5.99	0.21	7.73	0.26	9.84	0.28	11.42	0.40
100	5.85	0.22	8.62	0.27	11.85	0.32	13.86	0.38	17.56	0.42
150	9.31	0.31	14.34	0.38	19.30	0.47	24.66	0.52	31.10	0.56
200	12.89	0.42	19.87	0.51	28.78	0.59	36.83	0.68	46.31	0.74

Table 2. A comparison of the execution time of the primal-dual approach (PD) with the execution time of the SeDuMi solver for convex programs (CP). The execution times are computed in *seconds*. The values of the table correspond to $\alpha = 2$, $w_{\max} = 10$, $p = 1$. Each entry of the table is the average over 30 instances. The empty entries correspond to cases with $m = n$ and take time longer than 30 minutes each and are interrupted.

The results presented above motivated us to further explore the case $n = m$. For this reason, we performed more experiments for $m = 10, 20, 30, 40, 50$ and $n = m-5, m-4, \ldots, m+4, m+5$. The results of these experiments are shown in Fig. 2. The horizontal axis corresponds to the difference $m-n$, while the vertical



(a) (b)

Fig. 1. Parameters: $\alpha = 2$, $w_{\max} = 10$, $p = 1$. (a) The number of modifications of the dual variables made by the primal-dual algorithm if $n > m$. (b) A comparison of the execution times of the primal-dual algorithm and the SeDuMi solver for convex programs if $n > m$ ($m = 10$).

axis corresponds to the logarithm of the modifications of the dual variables made by our algorithm.

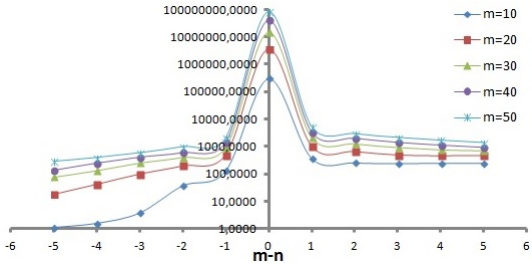


Fig. 2. Parameters: $\alpha = 2$, $w_{\max} = 10$, $p = 1$. The vertical axis represent the logarithm of the modifications of the dual variables made by the primal-dual algorithm.

We observe that the behavior of the primal-dual algorithm dramatically changes when $n = m$, while there is a much smaller perturbation when $n = m \pm 1$ and $n = m \pm 2$. In all other cases the number of modifications seems to increase linearly with the size of the instance. The problem with the case where $n = m$ probably occurs because in an optimal solution of a random instance almost all processors and jobs are tight, that is the total execution time of each processor and each job is equal to the deadline d. In other words, all β_i's and γ_j's are expected to be non-zero. The primal-dual algorithm, in each iteration "corrects" first the values of β_i's and then the values of γ_j's. As all of them are expected

to be non-zero in the optimal solution the required precision plays a significant role to the speed of the convergence of the algorithm.

6 Conclusions

We have presented a primal-dual algorithm in the general setting of convex programming and KKT conditions and we have proved that it converges to an optimal solution. In the direction of exploring the complexity of our algorithm, we performed simulations to observe its behavior when the size of the instance increases. Our experiments highlight the case in which the primal-dual algorithm has a problematic behavior, i.e., when $n = m$. In all other cases, and more interestingly in the case where $n > m$ that is closer to applications, the complexity of the algorithm seems to depend linearly on the size of the instance. An interesting open question remaining is whether our algorithm has a polynomial-time complexity. If not, the design of another algorithm based on the primal-dual paradigm that runs in polynomial time would be a challenging direction.

References

1. S. Albers. Energy-efficient algorithms. *Communications of ACM*, 53:86–96, 2010.
2. S. Albers. Algorithms for dynamic speed scaling. In *STACS'11*, pages 1–11, 2011.
3. E. Bampis, D. Letsios, and G. Lucarelli. Green scheduling, flows and matchings. In *ISAAC'12*, volume 7676 of *LNCS*, pages 106–115. Springer, 2012.
4. S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge Un. Press, 2004.
5. P. Brucker. *Scheduling algorithms (4th ed.)*. Springer, 2004.
6. N. R. Devanur, C. H. Papadimitriou, A. Saberi, and V. V. Vazirani. Market equilibrium via a primal-dual algorithm for a convex program. *Journal of the ACM*, 55(5), 2008.
7. J. Edmonds. Maximum matching and a polyhedron with $0, 1$-vertices. *Journal of Research of the National Bureau of Standards, Section B*, 69:125–130, 1965.
8. L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.
9. M. X. Goemans and D. P. Williamson. *The primal-dual method for approximation algorithms and its application to network design problems*, chapter 4, pages 144–191. PWS publishing company, 1997.
10. T. Gonzalez. A note on open shop preemptive schedules. *IEEE Transactions on Computers*, C-28:782–786, 1979.
11. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling. *Annals of Discrete Mathematics*, 5:287–326, 1979.
12. A. Gupta, R. Krishnaswamy, and K. Pruhs. Online primal-dual for non-linear optimization with applications to speed scaling. *CoRR*, abs/1109.5931, 2011.
13. H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
14. V. V. Vazirani. *Approximation algorithms*, chapter 12. Springer, 2001.
15. F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS'95*, pages 374–382, 1995.