

Corrigendum: Improved Results for Data Migration and Open Shop Scheduling

Rajiv Gandhi* Magnús M. Halldórsson† Guy Kortsarz‡
Hadas Shachnai §

Abstract

In [1], we gave an algorithm for the data migration and non-deterministic open shop scheduling problems in the minimum sum version, that was claimed to achieve a 5.06-approximation. Unfortunately, it was pointed to us by Maxim Sviridenko that the argument contained an unfounded assumption that has eluded all of its readers until now. We detail in this document how this error can be amended. A side effect is an improved approximation ratio of 4.96.

1 Introduction

The purpose of this note is to correct problems in our earlier paper [1].

We first define the data migration and open shop scheduling problems formally. It is easy to see that the latter is a special case of the former.

Data Migration Problem: Given is a graph $G = (V, E)$, where the vertices represent storage devices, and the edges correspond to data transmissions among the devices. Denote by $E(u)$ the set of edges incident on a vertex u . Each vertex v has weight w_v and processing time 0. Each edge e has a length, or processing time, p_e . Moreover, each edge e can be processed only after its release time r_e . All release times

*Department of Computer Science, Rutgers University, Camden, NJ 08102. E-mail: rajivg@camden.rutgers.edu. This work was partially supported by NSF awards CCF 0830569 and CCF 1143931.

†Reykjavik University, 101 Reykjavik, Iceland. E-mail: mmh@ru.is

‡Department of Computer Science, Rutgers University, Camden, NJ 08102. E-mail: guyk@camden.rutgers.edu. Partially supported by NSF grant number 0829959.

§Department of Computer Science, Technion, Haifa, 32000, Israel. Email: hadas@cs.technion.ac.il

and processing times are non-negative integers. The completion time of an edge is the time at which its processing is completed. Each vertex v can complete only after all the edges in $E(v)$ are completed. Since each vertex v has processing time 0, the completion time, C_v , of v is the latest completion time of any edge in $E(v)$. A crucial constraint is that two edges incident on the same vertex cannot be processed at the same time. The objective is to minimize $\sum_{v \in V} w_v C_v$.

Open Shop Scheduling Problem: Given is a set of jobs $\mathcal{J} = \{J_1, \dots, J_n\}$, to be scheduled on a set of machines $\mathcal{M} = \{M_1, \dots, M_m\}$. Each job J_j has a non-negative weight w_j ; also, J_j consists of a set of m operations $o_{j,1}, \dots, o_{j,m}$, with the corresponding processing times $p_{j,i}$, $1 \leq i \leq m$; the operation $o_{j,i}$ must be processed on the machine M_i . Each machine can process at most one operation at any time, and no two operations belonging to the same job can be processed simultaneously. The completion time C_j of each job J_j is the latest completion time of any of its operations. The objective is to minimize $\sum_{J_j \in \mathcal{J}} w_j C_j$.

Let us first clarify the erroneous assumption and its impact. Specifically, the algorithm was analyzed for two different wait functions, with the quality of the output parameterized by a coefficient λ_{e_u} for each vertex u . The parameter was based on properties of a specific edge e_u that depended on the running of the algorithm. The error made was to assume that the edge e_u would be the same edge in the two runs of the algorithm involving the different wait functions. Thus, while the main body of the analysis of the algorithm was all correct, the part about combining the two solution (starting with the unnumbered subsection titled ‘‘Combining the two solutions’’) was not well-founded. The error implies that the claimed 5.06-approximation ratio does not follow.

To amend this problem, we have modified the analysis so that instead of combining the result of runs of the algorithm with two wait functions, we use a single wait function that combines the two previous ones (with slight modification). Much of the tools derived in the analysis in [1] carry over, while the presentation is somewhat different. In some cases, we can actually streamline the arguments somewhat. Our presentation is self contained.

2 A Linear Programming Relaxation

We modify only slightly the linear programming relaxation of [1], strengthening the first constraint (1).

For an edge e (vertex v) let C_e (C_v) be the variable that represents the completion

time of e (resp., v) in the LP relaxation. For any set of edges $S \subseteq E$, let $p(S) = \sum_{e \in S} p_e$ and $p(S^2) = \sum_{e \in S} p_e^2$.

$$(LP) \quad \text{minimize} \quad \sum_{v \in V} w_v C_v$$

subject to:

$$C_e \geq r_e + p_e, \quad \forall e \in E \quad (1)$$

$$C_v \geq p(E(v)), \quad \forall v \in V \quad (2)$$

$$C_v \geq C_e, \quad \forall v \in V, e \in E(v) \quad (3)$$

$$\sum_{e \in S(v)} p_e C_e \geq \frac{p(S(v))^2 + p(S(v)^2)}{2}, \quad \forall v \in V, S(v) \subseteq E(v). \quad (4)$$

In [1] we proved a property of the LP that plays a crucial role in the analysis of our algorithm. Let $X(v, t_1, t_2) \subseteq E(v)$ denote the set of edges that complete in the time interval $(t_1, t_2]$ in the LP solution (namely, their fractional value belongs to this interval). Hall *et al.* [2] showed that $p(X(v, 0, t)) \leq 2t$. Lemma 2.1 gives a stronger property of a solution given by the above LP. Intuitively, our property states that if too many edges complete early, then the completion times of other edges must be delayed. For example, as a consequence of our property, for any $t > 0$ if $p(X(v, 0, t/2)) = t$ then $p(X(v, t/2, t)) = 0$, which means that no edges in $E(v) \setminus X(v, 0, t/2)$ complete before t in the LP solution.

Lemma 2.1 *Consider a vertex v and times $t_1 > 0$ and $t_2 \geq t_1$. If $p(X(v, 0, t_1)) = \lambda_1$ and $p(X(v, t_1, t_2)) = \lambda_2$, then λ_1 and λ_2 are related by*

$$\lambda_2 \leq t_2 - \lambda_1 + \sqrt{t_2^2 - 2\lambda_1(t_2 - t_1)}$$

The result of [2] follows from Lemma 2.1 by substituting $t_1 = 0$, $\lambda_1 = 0$, and $t_2 = t$.

Corollary 2.2 *For any vertex v and time $t \geq 0$, $p(X(v, 0, t)) \leq 2t$.*

3 Algorithm

The basic algorithm is unchanged from [1], but is given here for completeness.

Note that if an edge has processing time 0, it can be processed as soon as it is released, without consuming any time-steps. Hence, without loss of generality, we assume that the processing time of each edge is a positive integer.

The algorithm is parameterized by a *wait function* $W : E \rightarrow \mathcal{R}^+$. The idea is that each edge e must *wait* for W_e ($W_e \geq r_e$) time-steps before it can become *active* and start processing. Edges that have waited long enough are processed in non-decreasing order of their LP-values. Once an edge e becomes active, it remains active for p_e time-steps, after which it is *finished*. A not-yet-active edge can be *waiting* only if none of its neighboring edges is active; otherwise, it is said to be *delayed*. Thus, at any time, an edge is in one of four modes: *delayed*, *waiting*, *active*, or *finished*. The above specifies the operation of the algorithm.

The implementation of these rules is given in the pseudocode in Fig. 1. In prose, the algorithm operates as follows in each time-step t . Each edge e that had been active for p_e time-steps becomes *finished*, with a completion time of $t - 1$; if they had been active for less than p_e time-steps they stay active. We next consider the edges that are neither finished or active in non-decreasing order of LP-value: if edge e has no active incident edges and it has been waiting its share (for W_e time-steps), then it becomes *active*. The remaining edges are now set as *delayed*, if they have an active incident edge, or otherwise *waiting*. The algorithm operates until all vertices become finished.

Let $wait(e, t)$ denote the number of time-steps that e has waited until the end of time-step t . Let $Active(t)$ be the set of active edges during time-step t . Let \widetilde{C}_e (\widetilde{C}_u) be the completion time of edge e (vertex u) in our algorithm.

The algorithm in Fig. 1, implemented as is, would run in pseudo-polynomial time; however, it is easy to implement the algorithm in strongly polynomial time, by increasing t in each iteration by the smallest remaining processing time of an active edge.

For any vertex a (edge e), let C_a^* (C_e^*) be its value in an optimal LP solution. For $e = (a, b)$, let $S_e(a) = \{f \in E(a) | C_f^* \leq C_b^*\}$ be the set of edges incident on a that have LP-value no larger than C_b^* .

We use the following wait function on each edge $e = (a, b)$, which captures both of the wait functions used in [1]:

$$W_e = \lfloor \alpha \min\{C_a^*, C_b^*\} + \beta \max\{r_e, p(S_e(a)), p(S_e(b))\} \rfloor, \quad (5)$$

for some fixed $\alpha, \beta \geq 0$ to be determined. When the edges have release times, we additionally require that $\alpha + \beta \geq 1$, which ensures that an edge becomes active only after it is released. We have optimized of possible values of α and β to find the setting

```

SCHEDULE( $G = (V, E), W$ )
1   Solve the LP relaxation for the given instance optimally.
2    $t \leftarrow 0$ 
3    $Finished \leftarrow Active(t) \leftarrow \emptyset$ 
4   for each  $e \in E$  do
5        $wait(e, t) \leftarrow 0$ 
6   while ( $Finished \neq E$ ) do
7        $t \leftarrow t + 1$ 
8        $Active(t) \leftarrow \{e \mid e \in Active(t - 1) \text{ and } e \notin Active(t - p_e)\}$ 
9       for each edge  $e \in Active(t - 1) \setminus Active(t)$  do
10           $\widetilde{C}_e \leftarrow t - 1$ 
11           $Finished \leftarrow Finished \cup \{e\}$            //  $e$  is finished
12       for each edge  $e = (u, v) \in E \setminus (Active(t) \cup Finished)$ 
13          in non-decreasing order of LP-value do
14          if ( $Active(t) \cap (E(u) \cup E(v)) = \emptyset$ ) and ( $wait(e, t - 1) = W_e$ ) then
15               $Active(t) \leftarrow Active(t) \cup \{e\}$            //  $e$  is active
16          for each edge  $e = (u, v) \in E \setminus (Active(t) \cup Finished)$  do
17              if ( $Active(t) \cap (E(u) \cup E(v)) = \emptyset$ ) then
18                   $wait(e, t) \leftarrow wait(e, t - 1) + 1$        //  $e$  is waiting
19                  else  $wait(e, t) \leftarrow wait(e, t - 1)$        //  $e$  is delayed
20  return  $\widetilde{C}$ 

```

Figure 1: Algorithm for Data Migration

$\alpha = 0.77$ and $\beta = 0.28$.

4 Analysis

We first give an overview of our approach. We examine an arbitrary but fixed vertex u , comparing its completion time \widetilde{C}_u to its LP-value C_u^* , thus obtaining the same bound for the whole instance. This time is equivalently the completion time of the latest incident edge to complete, that we shall denote $e_u = (u, v)$.

Before e_u becomes active – and then executes to completion – it will need to wait and may get delayed. It is easy to bound the costs of waits and delays caused by other edges incident on u . The challenging part is to bound X_v , representing the delays caused by edges incident on v . Particularly problematic is the set $B_{e_u}(v)$ of

edges incident on v that complete before e_u but have larger LP-value than that of u . For this purpose, we focus on the role of the edge $l = (v, w)$ with the largest LP-value (so, lowest priority) among the edges in $B_{e_u}(v)$ (or in $S_{e_u}(v)$, if $B_{e_u}(v)$ is empty). The ratio $\rho = C_l^*/C_u^*$ is one key parameter of the analysis; the other is λ_{e_u} , which captures the relative contribution of edges incident on v with smaller LP-value than u to the delays incurred by e_u . We give two different bounds on X_v : a generic one utilizing Lemma 4.5, and the other specific to our chosen wait function that is obtained via bounds on W_l , the waiting time of the edge l .

Before starting the analysis, we formally introduce the additional notation used. Let u be an arbitrary fixed vertex and $e_u = (u, v)$ be the edge such that $\widetilde{C}_{e_u} = \widetilde{C}_u$. For any edge $e = (a, b)$, let $B_e(a) = \{f \in E(a) \mid C_f^* > C_b^* \text{ and } \widetilde{C}_f < \widetilde{C}_e\}$, i.e., edges in $E(a)$ that finish before e in our algorithm, but have LP-value greater than C_b^* . Recall that $S_e(a) = \{f \in E(a) \mid C_f^* \leq C_b^*\}$. Note that $e \in S_e(a)$. Let $X_e(a) = S_e(a) \cup B_e(a)$ and $\overline{X}_e(a) = X_e(a) \setminus \{e\}$. For shorthand, write $X_v = p(X_{e_u}(v))$ ($X_u = p(X_{e_u}(u))$) and $\overline{X}_v = p(\overline{X}_{e_u}(v)) = X_v - p_{e_u}$ ($\overline{X}_u = X_u - p_{e_u}$).

By constraint (3), we have $C_{e_u}^* \leq C_u^*$. Note that $X_{e_u}(u) = E(u)$, and by constraint (2), we have $X_u \leq C_u^*$.

Define λ_{e_u} by

$$p(S_{e_u}(v)) = \lambda_{e_u} C_u^*, \quad 0 \leq \lambda_{e_u} \leq 2 \quad (6)$$

The upper bound on λ_{e_u} follows from Corollary 2.2, with t denoting C_u^* and $X(v, 0, t)$ denoting $S_{e_u}(v) = \{f \in E(v) \mid C_f^* \leq C_u^*\}$. Let $\hat{\lambda}_u = \max(1, \lambda_{e_u})$.

Finally, we define $l = (v, w)$ to be the edge with the *largest LP-value* among the edges in $X_{e_u}(v)$, i.e., $C_l^* = \max_{f \in X_{e_u}(v)} \{C_f^*\}$. Note that l is in $B_{e_u}(v)$ unless $B_{e_u}(v)$ is empty. Also, let $\rho = C_l^*/C_u^*$. Note that $\rho > 1$, if $B_{e_u}(v)$ is not empty.

For ease of reference, we summarize in Table 1 the key notation used in the rest of this section.

The completion time of a vertex u equals the completion time of the last incident edge e_u to be completed. The latter can be bounded by the sum of three quantities: delays caused by other edges incident on u , delays caused by edges incident on the other endpoint of e_u (namely v), and the waiting time of e_u . In the following lemmas, we shall show that each of these quantities can be bounded by a constant factor of the LP-value of the vertex.

The following lemma appears in slightly different form as Lemma 5.1 in [1].

Lemma 4.1 $\widetilde{C}_u \leq W_{e_u} + C_u^* + X_v$.

Proof: Observe that when e_u is in delayed mode it must be that some edge in $\overline{X}_{e_u}(u) \cup$

$\overline{X_{e_u}(v)}$ is active. Recall that $p(X_{e_u}(u)) = X_u \leq C_u^*$. Hence, we have

$$\widetilde{C}_u = \widetilde{C}_{e_u} \leq W_{e_u} + \overline{X_u} + \overline{X_v} + p_{e_u} = W_{e_u} + X_u + \overline{X_v} \leq W_{e_u} + C_u^* + \overline{X_v}.$$

□

Symbol	Description
W_e	waiting time of edge e .
$C_e^* (C_v^*)$	value of variable $C_e (C_v)$ of edge e (vertex v) in the LP solution.
$p(Z)$	$\sum_{e \in Z} p_e$.
$E(u)$	set of edges incident on vertex u .
r_e	release time of edge e .
$\widetilde{C}_e (\widetilde{C}_u)$	completion time of edge e (vertex u) in our algorithm.
$e_u = (u, v)$	edge finishing last in our algorithm among edges in $E(u)$.
$S_e(a)$	$\{f \in E(a) C_f^* \leq C_b^*\}$
$B_e(a)$	$\{f \in E(a) C_f^* > C_b^* \text{ and } \widetilde{C}_f < \widetilde{C}_e\}$.
$X_e(u) (\overline{X_e(u)})$	$S_e(u) \cup B_e(u) (X_e(u) \setminus \{e\})$
$X_v (\overline{X_v})$	$p(X_{e_u}(v)) (p(\overline{X_{e_u}(v)}))$.
$l = (v, w)$	edge in $B_{e_u}(v)$ with the largest LP-value.
ρ	C_l^*/C_u^* .
$\lambda_{e_u} (\hat{\lambda}_u)$	$p(S_{e_u}(v))/C_u^* (\max(1, \lambda_{e_u}))$

Table 1: Summary of notation

Bounding the waiting time of the edge e_u is straightforward.

Lemma 4.2 *When using the wait function (5), $W_{e_u} \leq (\alpha + \beta \hat{\lambda}_u) C_u^*$.*

Proof: By constraints (1) and (3) in the LP, $r_{e_u} \leq C_{e_u}^* \leq C_u^*$, and from constraint (2), $p(S_{e_u}(u)) \leq C_u^*$. Also, recall from (6) that $p(S_{e_u}(v)) = \lambda_{e_u} C_u^*$, $0 \leq \lambda_{e_u} \leq 2$ and that $\hat{\lambda}_u = \max(1, \lambda_{e_u})$. Hence, from (5), we have that

$$W_{e_u} \leq \lfloor (\alpha + \beta \max\{1, \lambda_{e_u}\}) C_u^* \rfloor \leq (\alpha + \beta \hat{\lambda}_u) C_u^*.$$

□

When $B_{e_u}(v)$ is empty we obtain the following bound on \widetilde{C}_u .

Lemma 4.3 *If $B_{e_u}(v) = \emptyset$, then $\widetilde{C}_u \leq (\alpha + 2\beta + 3) C_u^*$.*

Proof: Since $B_{e_u}(v) = \emptyset$, it follows from (6) that $X_v = p(S_{e_u}(v)) \leq 2C_u^*$. Combining this with Lemmas 4.1 and 4.2 proves our claim. \square

In the remainder of the section we consider the case where $B_{e_u}(v) \neq \emptyset$, in which case $\rho > 1$. The main challenge in this case is to bound the contribution of X_v . We argue two upper bounds on X_v , one independent of the wait function and the other one specific to our wait function.

The following lemma extracts the essential wait function-independent part of Lemma 5.2 in [1]. Recall that $\rho = C_l^*/C_u^*$.

Lemma 4.4 *If $B_{e_u}(v) \neq \emptyset$, then $X_v \leq \left(\rho + \sqrt{\rho^2 - 2\lambda_{e_u}(\rho - 1)}\right) C_u^*$.*

Proof: Note that in the LP solution, each edge in $S_{e_u}(v)$ finishes at or before C_u^* and each edge in $B_{e_u}(v)$ finishes after C_u^* and at or before C_l^* . Thus, by substituting $t_1 = C_u^*$, $t_2 = C_l^* = \rho C_u^*$, $\lambda_1 = p(S_{e_u}(v)) = \lambda_{e_u} C_u^*$, and $\lambda_2 = p(B_{e_u}(v))$ in Lemma 2.1 we get

$$\begin{aligned} p(B_{e_u}(v)) &\leq \rho C_u^* - \lambda_{e_u} C_u^* + \sqrt{\rho^2 (C_u^*)^2 - 2\lambda_{e_u} C_u^* (\rho C_u^* - C_u^*)} \\ &= \left(\rho - \lambda_{e_u} + \sqrt{\rho^2 - 2\lambda_{e_u}(\rho - 1)}\right) C_u^*. \end{aligned}$$

The lemma then follows from (6), since $X_v = p(B_{e_u}(v)) + p(S_{e_u}(v))$. \square

We first need to argue a bound on the waiting time of l , the edge in $B_{e_u}(v)$ with the largest LP-value. It appeared implicitly within Lemmas 5.2 and 5.3 in [1].

Lemma 4.5 *If $B_{e_u}(v) \neq \emptyset$, then $W_l \leq W_{e_u} + C_u^* - 1$.*

Proof: Consider any time-step when l is in waiting mode. Since $l \in B_{e_u}(v)$, it is completed by the algorithm earlier than e_u , so e_u cannot be finished. By definition, e_u also cannot be active. Thus, it is either waiting or it is delayed. Since the scheduling is non-preemptive, e_u can only be delayed by edges g in $E(u)$ with $\widetilde{C}_g < \widetilde{C}_u$. All such edges are contained in $\overline{X_{e_u}}(u)$. So, the amount of time that e_u is delayed while l is waiting is at most $\overline{X}_u = X_u - p_{e_u} \leq X_u - 1 \leq C_u^* - 1$. \square

The second bound on X_v utilizes the particular wait function W_e used.

Lemma 4.6 *Assume $B_{e_u}(v) \neq \emptyset$. Then, when using the wait function of (5), $X_v \leq \left(\frac{1+\alpha(1-\rho)+\beta\lambda_u}{\beta}\right) C_u^*$.*

Proof: Let $g \in X_{e_u}(v)$. Then, by the definition of $l = (v, w)$ and constraint (3), $C_g^* \leq C_l^* \leq C_w^*$. Hence, $g \in S_l(v) = \{f \in E(v) | C_f^* \leq C_w^*\}$. Thus, $X_{e_u}(v) \subseteq S_l(v)$, and equivalently

$$X_v \leq p(S_l(v)) . \quad (7)$$

By (5), (3) and (7), respectively,

$$W_l \geq \lfloor \alpha \min\{C_v^*, C_w^*\} + \beta \cdot p(S_l(v)) \rfloor \geq \lfloor \alpha C_l^* + \beta \cdot p(S_l(v)) \rfloor \geq \alpha \rho C_u^* + \beta X_v - 1 . \quad (8)$$

By Lemmas 4.5 and 4.2,

$$W_l \leq (1 + \alpha + \beta \hat{\lambda}_u) C_u^* - 1 . \quad (9)$$

Combining (8) and (9), we get

$$\beta X_v \leq (1 + \alpha(1 - \rho) + \beta \hat{\lambda}_u) C_u^* . \quad (10)$$

Dividing through by β yields the lemma. \square

Finally, we combine the above bounds to argue the performance of the algorithm. The performance guarantee applies to all vertices simultaneously, in terms of their LP-values.

Theorem 4.7 *There exists a 4.96-approximation algorithm for the data migration problem, as well as for the open shop scheduling problem.*

Proof: Consider any vertex u , with e_u and ρ defined as before. Consider first the case $B_{e_u}(v) \neq \emptyset$. Combining Lemmas 4.1, 4.2, 4.4 and 4.6, we get

$$\begin{aligned} \widetilde{C}_u &\leq (1 + \alpha + \beta \hat{\lambda}_u) C_u^* + X_v \\ &\leq \left(1 + \alpha + \beta \hat{\lambda}_u + \min \left(\rho + \sqrt{\rho^2 - 2\lambda_{e_u}(\rho - 1)}, \frac{1 + \alpha(1 - \rho) + \beta \hat{\lambda}_u}{\beta} \right) \right) C_u^* . \end{aligned} \quad (11)$$

Choosing $\alpha = 0.77$ and $\beta = 0.28$, we obtain

$$\widetilde{C}_u \leq \left(1.77 + 0.28 \hat{\lambda}_u + \min \left(\rho + \sqrt{\rho^2 - 2\lambda_{e_u}(\rho - 1)}, 6.321 - 2.75\rho + \hat{\lambda}_u \right) \right) C_u^* .$$

We note that the expression is strictly decreasing for $\lambda_{e_u} \in [0, 1]$. It can be easily verified that the case $\lambda_{e_u} = 0$ results in a bound of $\widetilde{C}_u \leq 4.44 C_u^*$. Thus, we may assume that $\lambda_{e_u} = \hat{\lambda}_u = \lambda \in [1, 2]$. Also, recall that $\rho > 1$, and one can verify that

the case $\rho > 2.1$ results in a weaker bound than we claim. Thus, we may assume $\rho \in (1, 2.1)$. By trying all possible values for ρ and λ spaced 0.0001 apart, we find that the maximum value is attained when $\lambda = 1.509$ and $\rho = 1.842$, for a ratio of 4.9575. Given that the function grows at most linearly in both λ and ρ , we conclude that the absolute maximum differs by at most 0.0005, attaining a bound of $\widetilde{C}_u \leq 4.958 \cdot C_u^*$.

When $B_{e_u}(v) = \emptyset$, then with our choice of α and β , Lemma 4.3 gives $\widetilde{C}_u \leq 4.33C_u^*$. Thus, it holds unconditionally that for any vertex u , $\widetilde{C}_u \leq 4.958 \cdot C_u^*$, which implies the theorem. \square

Acknowledgment

We thank Maxim Sviridenko for pointing out the error and for providing valuable additional feedback.

References

- [1] R. Gandhi, M. M. Halldórsson, G. Kortsarz, and H. Shachnai. Improved Results for Data Migration and Open Shop Scheduling. *ACM TALG* 2(1):116–129 (2006).
- [2] L. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to Minimize Average Completion Time: Off-line and On-line Approximation Algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.
- [3] M. M. Halldórsson, G. Kortsarz, and H. Shachnai. Sum Coloring Interval Graphs and k -Claw Free Graphs with Applications for Scheduling Dependent Jobs. *Algorithmica*, 37:187–209, 2003.
- [4] M. Queyranne. Structure of a Simple Scheduling Polyhedron. *Mathematical Programming*, 58:263–285, 1993.