Fast Divide-and-Conquer Algorithms for Preemptive Scheduling Problems with Controllable Processing Times —A Polymatroid Optimization Approach—

Natasha V. Shakhlevich[*] Akiyoshi Shioura[†] Vitaly A. Strusevich[‡]

April 4, 2008

Abstract

We consider a variety of preemptive scheduling problems with controllable processing times on a single machine and on identical/uniform parallel machines, where the objective is to minimize the total compression cost. In this paper, we propose fast divide-and-conquer algorithms for these scheduling problems. Our approach is based on the observation that each scheduling problem we discuss can be formulated as a polymatroid optimization problem. We develop a novel divide-and-conquer technique for the polymatroid optimization problem and then apply it to each scheduling problem. We show that each scheduling problem can be solved in $\mathrm{O}(\mathrm{T}_{\mathrm{feas}}(n) \times \log n)$ time by using our divide-and-conquer technique, where n is the number of jobs and $\mathrm{T}_{\mathrm{feas}}(n)$ denotes the time complexity of the corresponding feasible scheduling problem with n jobs. This approach yields faster algorithms for most of the scheduling problems discussed in this paper.

1 Introduction

We consider a variety of preemptive scheduling problems with controllable processing times on a single machine and on identical/uniform parallel machines. In this paper, we propose fast divide-and-conquer algorithms for these scheduling problems.

Our Problems *Preemptive scheduling problems with controllable processing times* discussed in this paper are described as follows. We have n jobs, which are to be processed on m machines. The sets of jobs and machines are denoted by $N = \{1, 2, \ldots, n\}$ and by $M = \{1, 2, \ldots, m\}$, respectively. Each job j has *processing requirement* $\overline{p}(j)$. If $m = 1$, we have a *single* machine; otherwise we have m (≥ 2) *parallel* machines. The parallel machines are called *identical* if their speeds are equal; otherwise, the machines are called *uniform* and machine i has a speed s_i, so that processing a job j on machine i for τ time units reduces its overall processing requirement by $s_i\tau$.

[*]School of Computing, University of Leeds, Leeds LS2 9JT, U.K., ns@comp.leeds.ac.uk

[†]Graduate School of Information Sciences, Tohoku University, Sendai 980-8579, Japan, shioura@dais.is.tohoku.ac.jp.

[‡]Department of Mathematical Sciences, University of Greenwich, Old Royal Naval College, Park Row, London SE10 9LS, U.K., V.Strusevich@greenwich.ac.uk

In the processing of each job *preemption* is allowed, so that the processing of any job can be interrupted at any time and resumed later, possibly on another machine. No job is allowed to be processed on several machines at a time, and each machine processes at most one job at a time. Job j also has *release date* $r(j)$ and *due date* $d(j)$, and any piece of job j should be scheduled between the time interval $[r(j), d(j)]$.

Suppose that the processing requirement $\overline{p}(j)$ $(j \in N)$ cannot be feasibly scheduled on the machines. Then, we can reduce the processing requirement $\overline{p}(j)$ to $p(j)$ $(\leq \overline{p}(j))$ by paying the cost $w(j)(\overline{p}(j) - p(j))$ so that jobs can be feasibly scheduled. We here assume that the lower bound $\underline{p}(j)$ of the processing requirement $p(j)$ is given and $p(j) \geq \underline{p}(j)$ should be satisfied. The objective is to minimize the total cost $\sum_{j=1}^{n} w(j)(\overline{p}(j) - p(j))$ subject to the constraints that (i) processing requirement $p(j)$ $(j \in N)$ can be feasibly scheduled on m machines, and (ii) $\underline{p}(j) \leq p(j) \leq \overline{p}(j)$ $(j \in N)$. In this paper, we mainly consider an equivalent problem of maximizing $\sum_{j=1}^{n} w(j)p(j)$ under the same constraints (i) and (ii). We refer to [10] for comprehensive treatment of this problem.

Preemptive scheduling problem with controllable processing times is also known by different names with different interpretations. *Scheduling of imprecise computation tasks* (see, e.g., [2, 10, 19, 20, 21]; see also [17]) is an equivalent problem, where the portion $\overline{p}(j) - p(j)$ of job j is interpreted as the error of computation and $\sum_{j=1}^{n} w(j)(\overline{p}(j) - p(j))$ is regarded as the total weighted error. *The scheduling minimizing the weighted number of tardy task units* (see, e.g., [7, 11]) is equivalent to the special case with $\underline{p}(j) = 0$ for all job j, where the value $\overline{p}(j) - p(j)$ is regarded as the portion of the processing requirement which cannot be processed before the due date $d(j)$.

There are many kinds of preemptive scheduling problems with controllable processing times, depending on the setting of the underlying scheduling problems. In this paper, we consider three types of machines: a single machine, identical parallel machines, and uniform parallel machines. We also consider the cases where release/due dates of jobs are the same or arbitrary. We assume $r(j) = 0$ (resp., $d(j) = d$ (> 0)) for all $j \in N$ if all jobs have the same release dates (resp., due dates).

We denote each problem as {Single, Ide, Uni}-{SameR, ArbR}-{SameD, ArbD}. For example, Ide-SameR-ArbD denotes the the identical parallel machine scheduling problem with the same $r(j)$ and arbitrary $d(j)$. We note that the problem {Single, Ide, Uni}-SameR-ArbD is equivalent to {Single, Ide, Uni}-ArbR-SameD (see, e.g, [8, 15, 16]), and therefore need not be considered. Hence, we deal with nine problems in this paper.

Previous Results We review the current fastest algorithms for nine scheduling problems discussed in this paper. The summary is given in Table 1.

Single-SameR-SameD can be easily solved in $O(n)$ time. Janiak and Kovalyov [9] formulate Single-SameR-ArbD as a linear program and show that the linear program can be solved in $O(n \log n)$ time. For Single-ArbR-ArbD, Leung et al. [11] slightly improve the analysis of the $O(n^2)$-time greedy algorithm in [20] and obtain a better bound $O(n \log n + \kappa n)$, where κ is the number of distinct $w(j)$. Shih et al. [19] propose an $O(n \log n)$-time divide-and-conquer algorithm

Single	**Previous Results**	**This Paper**	**Feasible Schedule**
same $r(j)$ same $d(j)$	$O(n)$ (trivial) (best possible bound)	—	$O(n)$ (trivial)
same $r(j)$ arbitrary $d(j)$	$O(n \log n)$ [9] (best possible bound)	—	$O(n \log n)$ [13]
arbitrary $r(j)$ arbitrary $d(j)$	$O(n \log n + \kappa n)$ [11] $O(n \log n)$ (for $\mathbb{Z}$) [19]	$O(n \log n)$	$O(n \log n)$ [8] ($O(n)$ w/o sorting)
Identical	**Previous Results**	**This Paper**	**Feasible Schedule**
same $r(j)$ same $d(j)$	$O(n)$ [17] (best possible bound)	—	$O(n)$ [13]
same $r(j)$ arbitrary $d(j)$	$O(n^2(\log n)^2)$ [12]	$O(n \log m \log n)$	$O(n \log n)$ [15] ($O(n \log m)$ w/o sorting)
arbitrary $r(j)$ arbitrary $d(j)$	$O(n^2(\log n)^2)$ [12]	—	$O(n^2(\log n)^2)$ (cf. [2, 21])
Uniform	**Previous Results**	**This Paper**	**Feasible Schedule**
same $r(j)$ same $d(j)$	$O(mn + n \log n)$ [14]	$O(\min\{n \log n,$ $n+m \log m \log n\})$	$O(m \log m + n)$ [6]
same $r(j)$ arbitrary $d(j)$	$O(mn^3)$ [12]	$O(mn \log n)$	$O(mn + n \log n)$ [16] ($O(mn)$ w/o sorting)
arbitrary $r(j)$ arbitrary $d(j)$	$O(mn^3)$ [12]	—	$O(mn^3)$ [3]

Table 1: Summary of Our Results and Previous Results.
The time complexity with "w/o sorting" means the time complexity except for the time required for sorting input numbers of size $O(n)$ such as $w(j), r(j), d(j)$.

for Single-ArbR-ArbD, which works only for instances with the numbers $\overline{p}(j), \underline{p}(j), r(j), d(j)$ given by integers.

Ide-SameR-SameD can be formulated as the continuous knapsack problem by using the result of McNaughton [13], and therefore can be solved in $O(n)$ time (see [17]). McCormick [12] shows that Ide-ArbR-ArbD (and also Ide-SameR-ArbD) can be formulated as a parametric max flow problem, and applies the algorithm of Gallo et al. [5] to achieve the time complexity $O(n^3 \log n)$, which can be reduced to $O(n^2(\log n)^2)$ by using the balanced binary tree representation of time intervals as in Chung et al. [2, 21].

Uni-SameR-SameD can be solved by a greedy algorithm in $O(mn + n \log n)$ time [14]. McCormick [12] shows that Uni-ArbR-ArbD (and also Uni-SameR-ArbD) can be formulated as a parametric max flow problem on a bipartite network, and applies the algorithm of Ahuja et al. [1] to achieve the time complexity $O(mn^3)$.

Our Approach and Results The summary of our results is given in Table 1. Our approach is based on the observation that each of nine scheduling problems discussed in this paper can be formulated as a polymatroid optimization problem of the following form:

$$\begin{array}{lll}\text{(LP)} & \text{Maximize} & \displaystyle\sum_{j=1}^{n} w(j)p(j) \\ & \text{subject to} & p(Y) \leq \varphi(Y)\ (Y \in 2^N), \quad \underline{p}(j) \leq p(j) \leq \overline{p}(j)\ (j \in N),\end{array}$$

where $p(Y) = \sum_{j \in Y} p(j)$ and $\varphi : 2^N \to \mathbb{R}_+$ is a polymatroid rank function, i.e., a nondecreasing submodular function with $\varphi(\emptyset) = 0$. This observation is already made in [12, 17, 18] and used to show the validity of greedy algorithms for the scheduling problems. On the other hand, we use this observation in a different way; we develop a novel divide-and-conquer technique for the problem (LP), and apply it to the scheduling problems to obtain faster algorithms.

We define a function $\widetilde{\varphi} : 2^N \to \mathbb{R}$ by

$$\widetilde{\varphi}(X) = \min_{Y \in 2^N} \{\varphi(Y) + \overline{p}(X \setminus Y) - \underline{p}(Y \setminus X)\} \quad (X \in 2^N).$$

Then, $\widetilde{\varphi}$ is also a polymatroid rank funciton, and the set of maximal feasible solutions of (LP) is given as $\{p \in \mathbb{R}^n \mid p(Y) \leq \widetilde{\varphi}(Y)\ (Y \in 2^N),\ p(N) = \widetilde{\varphi}(N)\}$ (cf. [4, Sect. 3.1(b)]). Our divide-and-conquer technique is based on the following property, where the proof is given in Sect. 2:

Theorem 1.1. *Let $k \in N$ and $N_k = \{j \in N \mid w(j) \geq w(k)\}$. Suppose that $X_* \in 2^N$ satisfies*

$$\widetilde{\varphi}(N_k) = \varphi(X_*) + \overline{p}(N_k \setminus X_*) - \underline{p}(X_* \setminus N_k). \tag{1.1}$$

Then, there exists an optimal solution $q \in \mathbb{R}^n$ of the problem (LP) *satisfying $q(X_*) = \varphi(X_*)$, $q(j) = \overline{p}(j)$ $(j \in N_k \setminus X_*)$, and $q(j) = \underline{p}(j)$ $(j \in X_* \setminus N_k)$.*

By Theorem 1.1, the problem (LP) can be decomposed into two subproblems of similar structure, where the one is with respect to the variables $\{p(j) \mid j \in X_*\}$ and the other with respect to $\{p(j) \mid j \in N \setminus X_*\}$. Moreover, Theorem 1.1 shows that some of the variables can be fixed, which implies that each of the two subproblems contains at most $n/2$ non-fixed variables if we choose $k = n/2$. Hence, we can show that the depth of recursion is $\mathrm{O}(\log n)$ when this decomposition technique is applied recursively to (LP).

We also show that a subset $X_* \in 2^N$ satisfying (1.1) can be computed in $\mathrm{O}(\mathrm{T}_{\text{feas}}(n))$ time for each of the scheduling problem, where $\mathrm{T}_{\text{feas}}(n)$ denotes the time complexity for computing a feasible schedule with n jobs, except for the time required for sorting the input numbers (see Table 1 for the actual time complexity for comptuing a feasible schedule). This implies that each scheduling problem can be solved in $\mathrm{O}(\mathrm{T}_{\text{feas}}(n) \times \log n)$ time by using our divide-and-conquer technique. By applying this approach, we can obtain faster algorithms for four of the nine scheduling problems discussed in this paper (see Table 1).

Organization of This Paper In Sect. 2 we explain our divide-and-conquer algorithm for (LP). We then apply the divide-and-conquer technique to each scheduling problem in the following sections. We first give an $\mathrm{O}(n \log n)$-time algorithm for Single-ArbR-ArbD in Sect. 3. We show in Sect. 4 that Ide-SameR-ArbD and Uni-SameR-ArbD can be solved in $\mathrm{O}(n \log m \log n)$ time and $\mathrm{O}(mn \log n)$ time, respectively, in Sect. 4. Finally, two algorithms for Uni-SameR-SameD, which run in $\mathrm{O}(n \log n)$ time and $\mathrm{O}(n + m \log m \log n)$ time, respectively, are presented in Sect. 5. Proofs omitted in the main part of this paper are given in Appendix.

2 Divide-and-Conquer Technique for Polymatroid Optimization

We explain our divide-and-conquer technique for the problem (LP). The discussion in this section is based on basic properties of polymatroids and submodular polyhedra (see, e.g., [4]).

We, without loss of generality, assume that the weights $w(j)$ $(j \in N)$ are all distinct; this assumption can be easily fulfilled, e.g., by using perturbation. In addition, we suppose that a subset $F \subseteq N$ such that $\underline{p}(j) = \overline{p}(j)$ $(j \in F)$ is given, i.e, the variable $p(j)$ for each job $j \in F$ is already fixed. The set F is called a *fixed-job set*, and will be used in the divide-and-conquer algorithm. We denote by $\hat{n}$ the number of non-fixed variables in (LP), i.e., $\hat{n} = n - |F|$.

We first show how to decompose the problem (LP) into subproblems. Let $k \in N$ be an integer with $|N_k \setminus F| = \lfloor \hat{n}/2 \rfloor$, where $N_k = \{j \in N \mid w(j) \geq w(k)\}$. Suppose that (1.1) holds for some $X_* \in 2^N$. By Theorem 1.1, the problem (LP) can be decomposed into the following two subproblems of smaller size:

$$\begin{array}{lll}
\text{(SLP1)} & \text{Maximize} & \sum_{j \in X_*} w(j)p(j) \\
 & \text{subject to} & p(Y) \leq \varphi_1(Y) \ (Y \in 2^{X_*}), \\
 & & \underline{p}(j) \leq p(j) \leq \bar{p}(j) \ (j \in X_* \cap N_k), \quad p(j) = \underline{p}(j) \ (j \in X_* \setminus N_k), \\
\text{(SLP2)} & \text{Maximize} & \sum_{j \in N \setminus X_*} w(j)p(j) \\
 & \text{subject to} & p(Y) \leq \varphi_2(Y) \ (Y \in 2^{N \setminus X_*}), \\
 & & \underline{p}(j) \leq p(j) \leq \bar{p}(j) \ (j \in (N \setminus N_k) \setminus X_*), \quad p(j) = \overline{p}(j) \ (j \in N_k \setminus X_*),
\end{array}$$

where $\varphi_1 : 2^{X_*} \to \mathbb{R}$ and $\varphi_2 : 2^{N \setminus X_*} \to \mathbb{R}$ are defined as

$$\varphi_1(Y) = \varphi(Y) \quad (Y \in 2^{X_*}), \qquad \varphi_2(Y) = \varphi(Y \cup X_*) - \varphi(X_*) \quad (Y \in 2^{N \setminus X_*}).$$

Note that the subproblems (SLP1) and (SLP2) (and their corresponding scheduling problems) have a structure similar to that of the original problem (LP).

Lemma 2.1. *Suppose that $p_1 \in \mathbb{R}^{X_*}$ (resp., $p_2 \in \mathbb{R}^{N \setminus X_*}$) is an optimal solution of* (SLP1) *with $p_1(X_*) = \varphi_1(X_*)$ (resp.,* (SLP2) *with $p_2(N \setminus X_*) = \varphi_2(N \setminus X_*)$). Then, the direct sum $p = p_1 \oplus p_2 \in \mathbb{R}^n$ of p_1 and p_2 defined by*

$$(p_1 \oplus p_2)(j) = \begin{cases} p_1(j) & (j \in X_*), \\ p_2(j) & (j \in N \setminus X_*) \end{cases}$$

is an optimal solution of (LP) *with $p(N) = \varphi(N)$.*

The fixed-job sets for (SLP1) and (SLP2) are given by $F_1 = (F \cap X_*) \cup (X_* \setminus N_k)$ and $F_2 = (F \setminus X_*) \cup (N_k \setminus X_*)$, respectively. Since

$$|X_* \setminus F_1| \leq |N_k \setminus F| = \lfloor \hat{n}/2 \rfloor, \quad |(N \setminus X_*) \setminus F_2| \leq |(N \setminus N_k) \setminus F| = \lceil \hat{n}/2 \rceil, \tag{2.1}$$

the numbers of non-fixed variables in (SLP1) and in (SLP2) are at most half of that in (LP). This implies that the depth of recursion is $O(\log n)$ when this decomposition is applied recursively.

We then explain how to compute $X_* \in 2^N$ satisfying (1.1). We have

$$\widetilde{\varphi}(N_k) = -\underline{p}(N \setminus N_k) + \min_{X \in 2^N} \{\varphi(X) + \overline{p}(N_k \setminus X) + \underline{p}((N \setminus N_k) \setminus X)\}, \tag{2.2}$$

and the second term in the right-hand side of (2.2) is equal to the optimal value of the following problem (cf. [4, Sect. 3.1 (b)]):

$$\begin{array}{lll} \text{(ULP)} & \text{Maximize} & \displaystyle\sum_{j=1}^{n} p(j) \\ & \text{subject to} & p(X) \leq \varphi(X) \qquad (X \in 2^N), \\ & & 0 \leq p(j) \leq u(j) \qquad (j \in N), \end{array}$$

where

$$u(j) = \begin{cases} \overline{p}(j) & (j \in N_k), \\ \underline{p}(j) & (j \in N \setminus N_k). \end{cases} \tag{2.3}$$

The problem (ULP) is a special case of the problem (LP) where the objective function is unweighted, i.e., $w(j) = 1$ $(j \in N)$, and the lower bound of the variable $p(j)$ $(j \in N)$ is equal to zero, and therefore easier to solve. The scheduling problem corresponding to (ULP) is to maximize the sum of processing requirements under the upper bound constraint and the feasibility constraint that the processing requirements can be feasibly scheduled on machines, and can be solved in $\mathrm{O}(\mathrm{T}_{\mathrm{feas}}(n))$ time, in a similar way as the problem of computing a feasible schedule.

Let $q \in \mathbb{R}^n$ be an optimal solution of (ULP), and $X_* \in 2^N$ the unique maximal set with $q(X_*) = \varphi(X_*)$. It is shown in the following sections that such X_* can be computed in $\mathrm{O}(\mathrm{T}_{\mathrm{feas}}(n))$ time. By the optimality of q and submodularity of φ, we have

$$q(j) = \overline{p}(j) \ (j \in N_k \setminus X_*), \quad q(j) = \underline{p}(j) \ (j \in (N \setminus N_k) \setminus X_*)),$$

implying that X_* satisfies (1.1) since

$$\begin{aligned} \widetilde{\varphi}(N_k) &= -\underline{p}(N \setminus N_k) + q(N) \\ &= -\underline{p}(N \setminus N_k) + \{\varphi(X_*) + \overline{p}(N_k \setminus X_*) + \underline{p}((N \setminus N_k) \setminus X_*)\} \\ &= \varphi(X_*) + \overline{p}(N_k \setminus X_*) - \underline{p}(X_* \setminus N_k). \end{aligned}$$

Finally, we analyze the time complexity of our divide-and-conquer algorithm. Let $T(n, \hat{n})$ be the time complexity for solving the problem (LP) with n variables and $\hat{n}$ non-fixed variables, except for the time required for sorting input numbers. Then, we have

$$T(n, \hat{n}) = \mathrm{O}(\mathrm{T}_{\mathrm{feas}}(n)) + T(n_1, n_1') + T(n_2, n_2'),$$

where $n_1 + n_2 = n$, $n_1' \leq \min\{n_1, \lfloor \hat{n}/2 \rfloor\}$, and $n_2' \leq \min\{n_2, \lceil \hat{n}/2 \rceil\}$. By solving the recursive equation, we have $T(n, \hat{n}) = \mathrm{O}(\mathrm{T}_{\mathrm{feas}}(n) \times \log n)$.

Theorem 2.2. *Suppose that a subset $X_* \in 2^N$ satisfying* (1.1) *can be computed in* $\mathrm{O}(\mathrm{T}_{\mathrm{feas}}(n))$ *time. Then, the problem* (LP) *can be solved in* $\mathrm{O}(\mathrm{T}_{\mathrm{feas}}(n) \times \log n)$ *time.*

Finally, we give a proof of Theorem 1.1.

Proof of Theorem 1.1. Since the set of maximal feasible solutions of (LP) is given as $\{p \in \mathbb{R}^n \mid p(Y) \le \widetilde{\varphi}(Y)\ (Y \in 2^N),\ p(N)=\widetilde{\varphi}(N)\}$, the vector $p^* \in \mathbb{R}^n$ given by $p^*(j) = \widetilde{\varphi}(N_j) - \widetilde{\varphi}(N_{j-1})$ $(j = 1, 2, \ldots, n)$ is an optimal solution of (LP) (cf. [4, Sect. 3.1]). We show that the vector $q = p^*$ satisfies the conditions

$$q(X_*) = \varphi(X_*), \quad q(j) = \overline{p}(j)\ (j \in N_k \setminus X_*), \quad q(j) = \underline{p}(j)\ (j \in X_* \setminus N_k). \tag{2.4}$$

Since p^* is a feasible solution of the problem (LP), we have

$$p^*(X_*) \le \varphi(X_*), \quad p^*(j) \le \overline{p}(j)\ (j \in N_k \setminus X_*), \quad -p^*(j) \le -\underline{p}(j)\ (j \in X_* \setminus N_k). \tag{2.5}$$

By the definition of p^*, we have $p^*(N_k) = \widetilde{\varphi}(N_k) = \varphi(X_*) + \overline{p}(N_k \setminus X_*) - \underline{p}(X_* \setminus N_k)$, which, together with (2.5), implies that all the inequalities in (2.5) hold with equality. Hence, (2.4) follows. □

3 Single Machine with Arbitrary Release/Due Dates

We apply the divide-and-conquer technique in Sect. 2 to the problem Single-ArbR-ArbD. To describe the algorithm, we consider a restriction on the availability of the machine. Let $\tilde{I} = \{[g_k, g_{k+1}] \mid k = 1, 2, \ldots, 2n-1\}$ be a set of time intervals, where g_k is the k-th largest number in $\{r(j), d(j) \mid j \in N\}$. We are given a set of time intervals $I = \{[e_1, f_1], [e_2, f_2], \ldots, [e_\ell, f_\ell]\} \subseteq \tilde{I}$ such that the machine is available only in these time intervals, where $e_1 \le f_1 \le \cdots \le e_\ell \le f_\ell$. In addition, we are given a subset F of jobs (*fixed-job set*) such that $\underline{p}(j) = \overline{p}(j)$ for $j \in F$. We denote this variant of the problem Single-ArbR-ArbD by $\mathrm{P}(I, N, F)$. Any subproblem which appears during the recursive decomposition of the problem Single-ArbR-ArbD is of the form $\mathrm{P}(I, N, F)$; in particular, the original problem Single-ArbR-ArbD coincides with $\mathrm{P}(\tilde{I}, N, \emptyset)$.

The problem $\mathrm{P}(I, N, F)$ can be formulated as the problem (LP) with the polymatroid rank function $\varphi : 2^N \to \mathbb{R}$ given by

$$\varphi(X) = \sum\{f_k - e_k \mid 1 \le k \le \ell,\ [e_k, f_k] \subseteq [r(j), d(j)] \text{ for some } j \in X\}.$$

Let $k \in N$ be an integer with $|N_k \setminus F| = \lfloor \hat{n}/2 \rfloor$, where $\hat{n} = n - |F|$, and suppose that $X_* \in 2^N$ satisfies (1.1). Then, $\mathrm{P}(I, N, F)$ is decomposed into the subproblems $\mathrm{P}(I_1, N_1, F_1)$ and $\mathrm{P}(I_2, N_2, F_2)$, where

$$\begin{cases} I_1 = \{[e_k, f_k] \mid 1 \le k \le \ell,\ [e_k, f_k] \subseteq [r(j), d(j)] \text{ for some } j \in X_*\}, \\ N_1 = X_*,\ F_1 = (F \cap X_*) \cup (X_* \setminus N_k), \\ I_2 = I \setminus I_1,\ N_2 = N \setminus X_*,\ F_2 = (F \setminus X_*) \cup (N_k \setminus X_*). \end{cases}$$

In addition, we update $\underline{p}$ and $\overline{p}$ by

$$\overline{p}(j) := \underline{p}(j)\ (j \in X_* \setminus N_k), \qquad \underline{p}(j) := \overline{p}(j)\ (j \in N_k \setminus X_*). \tag{3.1}$$

We decompose the problem $\mathrm{P}(I, N, F)$ recursively in this way and compute an optimal solution.

We now explain how to compute $X_* \in 2^N$ satisfying (1.1) in $\mathrm{O}(n)$ time. It is assumed that the numbers $r(j), d(j)$ $(j \in N)$ and e_k, f_k $(k = 1, 2, \ldots, \ell)$ are already sorted. We firstly compute an optimal solution $q \in \mathbb{R}^n$ of the problem (ULP) corresponding to $\mathrm{P}(I, N, F)$, which can be done in $\mathrm{O}(\mathrm{T}_{\mathsf{feas}}(n)) = \mathrm{O}(n)$ time by using either of the algorithms in [7, 20]. Then, we compute a partition $\{N_0, N_1, \ldots, N_v\}$ of N such that $q(N_h) = \max_{j \in N_h} d(j) - \min_{j \in N_h} r(j)$ $(h = 1, 2, \ldots, v)$ and that $N \setminus N_0$ is maximal under this condition, which can be done in $\mathrm{O}(n)$ time. Since $\max_{j \in N_h} d(j) - \min_{j \in N_h} r(j) = \varphi(N_h)$ $(h = 1, \ldots, v)$, the set $X_* = N \setminus N_0$ is the unique maximal set with $q(X_*) = \varphi(X_*)$. Hence, Theorem 2.2 implies the following result.

Theorem 3.1. *The problem* $\mathrm{P}(I, N, F)$ *can be solved in* $\mathrm{O}(n \log n)$ *time. In particular, the problem* Single-ArbR-ArbD *can be solved in* $\mathrm{O}(n \log n)$ *time.*

It should be mentioned that our algorithm for Single-ArbR-ArbD is similar to the divide-and-conquer algorithm by Shih et al. [19], but the two algorithms are based on different ideas. Indeed, the algorithm in [19] works only for instances with the numbers $\overline{p}(j), \underline{p}(j), r(j), d(j)$ given by integers, while ours can be applied to any problem with real numbers.

4 Identical Parallel Machines with the Same Release Dates and Different Due Dates

We apply the divide-and-conquer technique in Sect. 2 to the problem Ide-SameR-ArbD. To describe the algorithm, we consider a restriction on the availability of the machines. Suppose that we are given numbers b_i $(i \in M)$ and c such that machine i is available in the time interval $[b_i, c]$. In addition, we are given a subset F of jobs (*fixed-job set*) such that $\underline{p}(j) = \overline{p}(j)$ for $j \in F$. We denote this variant of the problem Ide-SameR-ArbD by $\mathrm{P}(m, B, c, N, F)$, where $B = \{b_i \mid i \in M\}$. Any subproblem which appears during the recursive decomposition of the problem Ide-SameR-ArbD is of the form $\mathrm{P}(m, B, c, N, F)$; in particular, the original problem Ide-SameR-ArbD is the case where $b_1 = \cdots = b_m = 0$, $c = \max_{j \in N} d(j)$, and $F = \emptyset$.

The problem $\mathrm{P}(m, B, c, N, F)$ can be formulated as the problem (LP) with the polymatroid rank function $\varphi : 2^N \to \mathbb{R}$ given by

$$\varphi(X) = \sum_{i=1}^{m} \max\{\min\{d(i), c\} - b_i, 0\},$$

where we assume that $b_1 \leq b_2 \leq \cdots \leq b_m$ and that $d(i)$ is the i-th largest number in $\{d(j) \mid j \in N\}$ for $i = 1, \ldots, m$. Let $k \in N$ be an integer with $|N_k \setminus F| = \lfloor \hat{n}/2 \rfloor$, where $\hat{n} = n - |F|$, and suppose that $X_* \in 2^N$ satisfies (1.1). Then, the problem $\mathrm{P}(m, B, c, N, F)$ is decomposed into the subproblems $\mathrm{P}(m_1, B_1, c_1, N_1, F_1)$ and $\mathrm{P}(m_2, B_2, c_2, N_2, F_2)$, where

$$\begin{cases} m_1 = \min\{m, |X_*|\},\ B_1 = \{b_1, b_2, \ldots, b_{m_1}\},\ c_1 = \min\{c, d(j_{m_1})\}, \\ N_1 = X_*,\ F_1 = (F \cap X_*) \cup (X_* \setminus N_k), \\ m_2 = m,\ B_2 = \{d(j_1), \ldots, d(j_{m_1}), b_{m_1+1}, \ldots, b_m\},\ c_2 = c, \\ N_2 = N \setminus X_*,\ F_2 = (F \setminus X_*) \cup (N_k \setminus X_*), \end{cases}$$

where we assume that $\{j_1, j_2, \ldots, j_{m_1}\} \subseteq X_*$ and that $d(j_i)$ is the i-th largest number in $\{d(j) \mid j \in X_*\}$ for $i = 1, \ldots, m_1$. In addition, we update $\underline{p}$ and $\overline{p}$ by (3.1).

Suppose that $p_1 \in \mathbb{R}^{X_*}$ (resp., $p_2 \in \mathbb{R}^{N \setminus X_*}$) is an optimal solution of $\mathrm{P}(m_1, B_1, c_1, N_1, F_1)$ (resp., $\mathrm{P}(m_2, B_2, c_2, N_2, F_2)$). Then, the vector $p_* \in \mathbb{R}^n$ defined by

$$p_*(j) = \begin{cases} p_1(j) + \max\{0, d(j) - d(j_{m_1})\} & (j \in X_*), \\ p_2(j) & (j \in N \setminus X_*) \end{cases}$$

is an optimal solution of $\mathrm{P}(m, B, c, N, F)$.

We now explain how to compute $X_* \in 2^N$ satisfying (1.1) in $\mathrm{O}(n \log m)$ time. It is assumed that the numbers $d(j)$ $(j \in N)$ are already sorted. By using a slight modification of the algorithm by Sahni [15], we can compute an optimal solution $q \in \mathbb{R}^n$ of the problem (ULP) corresponding to $\mathrm{P}(m, B, c, N, F)$ in $\mathrm{O}(\mathrm{T}_{\mathrm{feas}}(n)) = \mathrm{O}(n \log m)$ time. Then, we compute the unique maximal set $X_* \in 2^N$ with $q(X_*) = \varphi(X_*)$. Using the following simple observations, we can find such X_* in $\mathrm{O}(n \log m)$ time.

Lemma 4.1.
(i) $\{j \in N \mid p(j) < u(j)\} \subseteq X_*$.
(ii) *Let* $j \in X_*$ *and* $j' \in N \setminus \{j\}$. *Suppose that there exists a time interval* $[e, f]$ *satisfying the following conditions:* $[e, f] \subseteq [r(j), d(j)]$, *any portion of job* j *is not processed on* $[e, f]$, *and some portion of job* j' *is processed on* $[e, f]$. *Then, we have* $j' \in X_*$.

Theorem 4.2. *The problem* $\mathrm{P}(m, B, c, N, F)$ *can be solved in* $\mathrm{O}(n \log m \log n)$ *time. In particular, the problem* Ide-SameR-ArbD *can be solved in* $\mathrm{O}(n \log m \log n)$ *time.*

We can solve the problem Uni-SameR-ArbD in a similar way as Ide-SameR-ArbD by using the algorithm of Sahni and Cho [16]. The details of the proof is omitted.

Theorem 4.3. *The problem* Ide-SameR-ArbD *can be solved in* $\mathrm{O}(mn \log n)$ *time.*

5 Uniform Parallel Machines with the Same Release/Due Dates

We apply the divide-and-conquer technique in Sect. 2 to the problem Uni-SameR-SameD. For the description of the algorithm, we consider the problem Uni-SameR-SameD with a subset F ob jobs (*fixed-job set*) such that $\underline{p}(j) = \overline{p}(j)$ for $j \in F$. We denote this problem by $\mathrm{P}(M, N, F)$, where M and N denote the sets of machines and jobs, respectively. Note that $\mathrm{P}(M, N, \emptyset)$ coincides with the original problem Uni-SameR-SameD. It is assumed that the speed of machines are already sorted and satisfy $s_1 \geq s_2 \geq \cdots \geq s_m$.

5.1 The First Algorithm

The problem $\mathrm{P}(M, N, F)$ can be formulated as (LP) with the polymatroid rank function $\varphi : 2^N \to \mathbb{R}$ given by $\varphi(X) = dS_{\min\{m, |X|\}}$ $(X \in 2^N)$, where $S_h = \sum_{i=1}^{h} s_i$ $(h = 1, \ldots, m)$. It can be decomposed (or reduced) into subproblems of smaller size, as follows. We assume that the numbers $\{\overline{p}(j) \mid j \in N\} \cup \{\underline{p}(j) \mid j \in N\}$ is already sorted.

The next property is a direct application of Theorem 1.1 to the problem $\mathrm{P}(M, N, F)$.

Lemma 5.1. *Let $k \in N$, and suppose that $X_* \in 2^N$ satisfies* (1.1). *Then, there exists an optimal solution $q \in \mathbb{R}^n$ of the problem $\mathrm{P}(M, N, F)$ satisfying the following properties, where $h = |X_*|$:*
(i) *if $h < m$, then $q(X_*) = dS_h$, $q(j) = \overline{p}(j)$ $(j \in N_k \setminus X_*)$, $q(j) = \underline{p}(j)$ $(j \in X_* \setminus N_k)$.*
(ii) *If $h \geq m$, then $q(N) = dS_m$ and $q(j) = \underline{p}(j)$ $(j \in N \setminus N_k)$.*

Let $k \in N$ be an integer with $|N_k \setminus F| = \lfloor \hat{n}/2 \rfloor$, where $\hat{n} = n - |F|$, and suppose that $X_* \in 2^N$ satisfies (1.1). Such X_* can be computed in $\mathrm{O}(n)$ time by Lemma 5.2 given below.

Lemma 5.2. *Suppose that the sorted list of the numbers $\mathcal{P} \equiv \{\overline{p}(j) \mid j \in N\} \cup \{\underline{p}(j) \mid j \in N\}$ is given. For any $X \in 2^N$, we can compute the value of $\widetilde{\varphi}(X)$ and a set $Y_* \in 2^N$ with $\widetilde{\varphi}(X) = \varphi(Y_*) + \overline{p}(X \setminus Y_*) - \underline{p}(Y_* \setminus X)$ in $\mathrm{O}(n)$ time.*

Let $h = |X_*|$. If $h < m$, then the problem $\mathrm{P}(M, N, F)$ can be decomposed into the following two subproblems $\mathrm{P}(M_1, N_1, F_1)$ and $\mathrm{P}(M_2, N_2, F_2)$, where

$$\begin{cases} M_1 = \{1, 2, \ldots, h\}, \quad N_1 = X_*, \quad F_1 = (F \cap X_*) \cup (X_* \setminus N_k), \\ M_2 = M \setminus M_1, \quad N_2 = N \setminus X_*, \quad F_2 = (F \setminus X_*) \cup (N_k \setminus X_*). \end{cases}$$

In addition, we update $\underline{p}$ and $\overline{p}$ by (3.1). Before solving the subproblems, we sort the numbers $\mathcal{P}_1 \equiv \{\overline{p}(j) \mid j \in N_1\} \cup \{\underline{p}(j) \mid j \in N_1\}$ and $\mathcal{P}_2 \equiv \{\overline{p}(j) \mid j \in N_2\} \cup \{\underline{p}(j) \mid j \in N_2\}$, which can be done in $\mathrm{O}(n)$ time. Hence, the decomposition can be done in $\mathrm{O}(n)$ time.

If $h \geq m$, then $\mathrm{P}(M, N, F)$ can be reduced to the subproblem $\mathrm{P}(M_1, N_1, F_1)$, where $M_1 = M$, $N_1 = N$, and $F_1 = F \cup (N \setminus N_k)$. In addition, we update $\underline{p}$ by $\overline{p}(j) := \underline{p}(j)$ $(j \in N \setminus N_k)$. Hence, the reduction can be done in $\mathrm{O}(n)$ time as well.

The following result follows from Theorem 2.2 and the discussion above.

Theorem 5.3. *The problem $\mathrm{P}(M, N, F)$ can be solved in $\mathrm{O}(n \log n)$ time by the first algorithm. In particular, the problem* Uni-SameR-SameD *can be solved in $\mathrm{O}(n \log n)$ time.*

We note that the time complexity of the first algorithm is optimal for Uni-SameR-SameD when $n = \mathrm{O}(m)$ since sorting the speeds of m machines requires $\mathrm{O}(m \log m)$ time.

5.2 The Second Algorithm

The running time of the first algorithm is dominated by the time for sorting the numbers in $\mathcal{P}$. To reduce the time complexity, we modify the first algorithm by using the information of the fixed-job set, so that it does not require the sorted list. We assume that the $\min\{m, |F|\}$ largest numbers in $\{\overline{p}(j) \mid j \in F\}$ and the number $\overline{p}(F)$ are given in advance.

Lemma 5.4. *Suppose that the $\min\{m, |F|\}$ largest numbers in $\{\overline{p}(j) \mid j \in F\}$ and the number $\overline{p}(F) = \sum_{j \in F} \overline{p}(j)$ are given. For any $X \in 2^N$, we can compute the value of $\widetilde{\varphi}(X)$ and a set $Y_* \in 2^N$ with $\widetilde{\varphi}(X) = \varphi(Y_*) + \overline{p}(X \setminus Y_*) - \underline{p}(Y_* \setminus X)$ in $\mathrm{O}((n - |F|) + m \log m)$ time.*

Hence, we can compute $X_* \in 2^N$ satisfying (1.1) in $\mathrm{O}((n - |F|) + m \log m)$ time. Using the set X_* we decompose (or reduce) the problem $\mathrm{P}(M, N, F)$ into subproblems in the same way as the first algorithm.

If $|X_*| < m$, then the problem $\mathrm{P}(M, N, F)$ can be decomposed into the two subproblems $\mathrm{P}(M_1, N_1, F_1)$ and $\mathrm{P}(M_2, N_2, F_2)$. The second subproblem $\mathrm{P}(M_2, N_2, F_2)$ is solved recursively by the second algorithm, while the first subproblem $\mathrm{P}(M_1, N_1, F_1)$ is solved by the first algorithm in $\mathrm{O}(|M_1| \log |M_1|) = \mathrm{O}(m \log m)$ time. Before solving $\mathrm{P}(M_2, N_2, F_2)$, we compute the $\min\{m, |F_2|\}$ largest numbers in $\{\overline{p}(j) \mid j \in F_2\}$ and the number $\overline{p}(F_2)$, which can be done in $\mathrm{O}((n - |F|) + m \log m)$ time.

If $|X_*| \geq m$, the problem $\mathrm{P}(M, N, F)$ is reduced to the subproblem $\mathrm{P}(M_1, N_1, F_1)$, which is recursively solved by the second algorithm. Before solving the subproblem, we compute the $\min\{m, |F_1|\}$ largest numbers in $\{\overline{p}(j) \mid j \in F_1\}$ and the number $\overline{p}(F_1)$, which can be done in $\mathrm{O}((n - |F|) + m \log m)$ time as well.

Let $T_2(m, n, \hat{n})$ denote the running time of the second algorithm for $\mathrm{P}(M, N, F)$. Then, the following recursive formula holds:

$$T_2(m, n, \hat{n}) = \begin{cases} \mathrm{O}(m \log m) & (\text{if } \hat{n} \leq 1), \\ \mathrm{O}(\hat{n} + m \log m) + T_2(|M_2|, |N_2|, |N_2| - |F_2|) & (\text{if } \hat{n} \geq 2,\ |X_*| < m), \\ \mathrm{O}(\hat{n} + m \log m) + T_2(|M_1|, |N_1|, |N_1| - |F_1|) & (\text{if } \hat{n} \geq 2,\ |X_*| \geq m). \end{cases}$$

Note that $|N_2| - |F_2| \leq \lceil \hat{n}/2 \rceil$ and $|N_1| - |F_1| \leq \lfloor \hat{n}/2 \rfloor$ by (2.1). Hence, we have $T_2(m, n, \hat{n}) = O(\hat{n} + m \log m \log \hat{n})$. As a preprocessing, we need to compute the $\min\{m, |F|\}$ largest numbers in $\{\overline{p}(j) \mid j \in F\}$ and the number $\overline{p}(F)$, which requires $\mathrm{O}(n + m \log m)$ time. Hence, the following result holds.

Theorem 5.5. *The problem* $\mathrm{P}(M, N, F)$ *can be solved in* $\mathrm{O}(n + m \log m \log n)$ *time by the second algorithm. In particular,* Uni-SameR-SameD *can be solved in* $\mathrm{O}(n + m \log m \log n)$ *time.*

References

[1] R. K. Ahuja, J. B. Orlin, C. Stein, R. E. Tarjan, Improved algorithms for bipartite network flow, *SIAM Journal on Computing* 23 (1994), 906–933.

[2] J. Y. Chung, W.-K. Shih, J. W. S. Liu, and D. W. Gillies, Scheduling imprecise computations to minimize total error, *Microprocessing and Microprogramming* 27 (1989), 767–774.

[3] A. Federgruen and H. Groenevelt, Preemptive scheduling of uniform machines by ordinary network flow techniques, *Management Science* 32 (1986), 341–349.

[4] S. Fujishige, Submodular Functions and Optimization, Second Edition, Annals of Discrete Mathematics 58, Elsevier, 2005.

[5] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan, A fast parametric maximum flow algorithm and applications, *SIAM Journal on Computing* 18 (1989), 30–55.

[6] T. F. Gonzales and S. Sahni, Preemptive scheduling of uniform processor systems, *Journal of ACM* 25 (1978), 92–101.

[7] D. S. Hochbaum and R. Shamir, Minimizing the number of tardy job unit under release time constraints, *Discrete Applied Mathematics* 28 (1990), 45–57.

[8] W. Horn, Some simple scheduling algorithms, *Naval Res. Logist. Quat.* 21 (1974), 177–185.

[9] A. Janiak and M.Y. Kovalyov, Single machine scheduling with deadlines and resource dependent processing times, *European Journal of Operational Research* 94 (1996), 284–291.

[10] J. Y.-T. Leung, Minimizing total weighted error for imprecise computation tasks and related problems, Chapter 34 of *Handbook of Scheduling* (ed.: J. Y-T. Leung), Chapman & Hall, 2004, 34-1–34-16.

[11] J. Y.-T. Leung, V. K. M. Yu, and W.-D. Wei, Minimizing the weighted number of tardy task units, *Discrete Applied Mathematics* 51 (1994), 307–316.

[12] S. T. McCormick, Fast algorithms for parametric scheduling come from extensions to parametric maximum flow, *Operations Research* 47 (1999), 744–756.

[13] R. McNaughton, Scheduling with deadlines and loss functions, *Management Science* 12 (1959), 1–12.

[14] E. Nowicki and S. Zdrzałka, A bicriterion approach to preemptive scheduling of parallel machines with controllable job processing times, *Discrete Appl. Math.* 63 (1995), 237–256.

[15] S. Sahni, Preemptive scheduling with due dates, *Operations Research* 27 (1979), 925–934.

[16] S. Sahni and Y. Cho, Scheduling independent tasks with due times on a uniform processor system, *Journal of the ACM* 27 (1980), 550–563.

[17] N. V. Shakhlevich and V. A. Strusevich, Pre-emptive scheduling problmes with controllable processing times, *Journal of Scheduling* 8 (2005), 233–253.

[18] N. V. Shakhlevich and V. A. Strusevich, Preemptive scheduling on uniform parallel machines with controllable job processing times, *Algorithmica*, to appear.

[19] W.-K. Shih, C.-R. Lee, and C.-H. Tang, A fast algorithm for scheduling imprecise computations with timing constraints to minimize weighted error, *Proceedings of the 21th IEEE Real-Time Systems Symposium (RTSS2000)* (2000), 305–310.

[20] W.-K. Shih, J. W. S. Liu, and J.-Y. Chung, Algorithms for scheduling imprecise computations with timing constraints, *SIAM Journal on Computing* 20 (1991), 537–552.

[21] W.-K. Shih, J. W. S. Liu, J.-Y. Chung, and D. W. Gillies, Scheduling tasks with ready times and deadlines to minimize average error, *ACM SIGOPS Operating Systems Review* 23 (1989), 14–28.

Appendix

A Proofs

A.1 Proof of Lemma 2.1

Lemma 2.1 is immediate from the following proposition and the fact that there exists an optimal solution $q \in \mathbb{R}^n$ of (LP) with $q(X_*) = \varphi(X_*)$.

Proposition A.1 (cf. [4, Lemma 3.1]). *Suppose that $p_1 \in \mathbb{R}^{X_*}$ (resp., $p_2 \in \mathbb{R}^{N \setminus X_*}$) is a feasible solution of* (SLP1) *with $p_1(X_*) = \varphi_1(X_*)$ (resp.,* (SLP2) *with $p_2(N \setminus X_*) = \varphi_2(N \setminus X_*)$). Then, the direct sum $p = p_1 \oplus p_2 \in \mathbb{R}^n$ of p_1 and p_2 defined by*

$$(p_1 \oplus p_2)(j) = \begin{cases} p_1(j) & (j \in X_*), \\ p_2(j) & (j \in N \setminus X_*) \end{cases}$$

is a feasible solution of (LP) *with $p(N) = \varphi(N)$. Conversely, for any feasible solution $p \in \mathbb{R}^n$ of* (LP) *satisfying $p(X_*) = \varphi(X_*)$, the restriction of p on X_* (resp., on $N \setminus X_*$) yields a feasible solution of* (SLP1) *with $p_1(X_*) = \varphi_1(X_*)$ (resp.,* (SLP2) *with $p_2(N \setminus X_*) = \varphi_2(N \setminus X_*)$).*

A.2 Proof of Lemma 5.2

Recall that the function $\varphi : 2^N \to \mathbb{R}$ is given by $\varphi(Y) = dS_{\min\{m,|Y|\}}$ $(Y \in 2^N)$ and the function value of $\varphi(Y)$ depends only on the cardinality of Y. For $i = 1, 2, \ldots, m-1$ we denote $\mathcal{F}_i = \{Y \in 2^N \mid |Y| = i\}$. Since $\underline{p}$ and $\overline{p}$ are nonnegative vectors, it follows from (2.2) that

$$\begin{aligned} \widetilde{\varphi}(X) &= -\underline{p}(N \setminus N_k) + \min_{Y \in 2^N} \{\varphi(Y) + \overline{p}(X \setminus Y) + \underline{p}((N \setminus X) \setminus Y)\} \\ &= -\underline{p}(N \setminus X) + \min\left[\varphi(N), \min_{Y \in \cup_{i=1}^{m-1} \mathcal{F}_i} \{\varphi(Y) + \overline{p}(X \setminus Y) + \underline{p}((N \setminus X) \setminus Y)\}\right] \\ &= -\underline{p}(N \setminus X) + \min\left[dS_m, \min_{0 \le i \le m-1} \left\{dS_i + \min_{Y \in \mathcal{F}_i} \{\overline{p}(X \setminus Y) + \underline{p}((N \setminus X) \setminus Y)\}\right\}\right] \\ &= -\underline{p}(N \setminus X) + \min\Bigg[dS_m, \min_{0 \le i \le m-1} \Bigg\{dS_i + \overline{p}(X) + \underline{p}(N \setminus X) \\ &\qquad - \max_{Y \in \mathcal{F}_i} \{\overline{p}(X \cap Y) + \underline{p}((N \setminus X) \cap Y)\}\Bigg\}\Bigg]. \end{aligned} \tag{A.1}$$

For $i = 1, 2, \ldots, m$, we denote by α_i the i-th largest number in the set $\mathcal{P}_X \equiv \{\overline{p}(j) \mid j \in X\} \cup \{\underline{p}(j) \mid j \in N \setminus X\}$. By using the sorted list of $\mathcal{P}$, the values $\alpha_1, \alpha_2, \ldots, \alpha_m$ can be computed in O(n) time. Then, we have

$$\max_{Y \in \mathcal{F}_i} \{\overline{p}(X \cap Y) + \underline{p}((N \setminus X) \cap Y)\} = \sum_{k=1}^{i} \alpha_k.$$

Hence, the value $\widetilde{\varphi}(X)$ can be computed in O(n) time by using the formula above. If the minimum in the RHS of (A.1) is attained by dS_m, then we can choose $Y_* = N$; otherwise, we can choose $Y_* = \{j_1, j_2, \ldots, j_k\}$, where $j_k \in N$ $(k = 1, 2, \ldots, m)$ is the number with $\alpha_k = \underline{p}(j_k)$ or $\alpha_k = \overline{p}(j_k)$.

A.3 Proof of Lemma 5.4

We prove the claim in a similar way as that for Lemma 5.2. Since

$$\mathcal{P}_X = \{\overline{p}(j) \mid j \in X \setminus F\} \cup \{\underline{p}(j) \mid j \in (N \setminus X) \setminus F\} \cup \{\overline{p}(j) \mid j \in F\}$$

and $(X \setminus F) \cup ((N \setminus X) \setminus F) \subseteq N \setminus F$, the m largest numbers in the set $\mathcal{P}_X$ can be obtained in $\mathrm{O}(|N \setminus F| + m \log m)$ time. We can also compute the value $\overline{p}(X) + \underline{p}(N \setminus X)$ in $\mathrm{O}(|N \setminus F|)$ time.