EasyChair Preprint

№ 13471

A 1.25(1+ε)-Approximation Algorithm for Scheduling with Rejection Costs Proportional to Processing Times

Olivier Beaumont, Rémi Bouzel, Lionel Eyraud-Dubois, Esragul Korkmaz, Laercio Pilla and Alexandre Van Kempen

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

May 29, 2024

A $1.25(1+\epsilon)$-Approximation Algorithm for Scheduling with Rejection Costs Proportional to Processing Times

Olivier Beaumont[1,2][0000−0003−2741−6228], Rémi Bouzel[3], Lionel Eyraud-Dubois[1,2][0000−0003−2475−3309], Esragul Korkmaz[1,2], Laercio Pilla[2,1][0000−0003−0997−586X], and Alexandre Van Kempen[3]

[1] Inria Center of the University of Bordeaux, France firstname.lastname@inria.fr
[2] LaBRI, UMR 5800, Talence, France
[3] Qarnot Computing, Montrouge, France
firstname.lastname@qarnot-computing.com

Abstract. We address an offline job scheduling problem where jobs can either be processed on a limited supply of energy-efficient machines, or offloaded to energy-inefficient machines (with an unlimited supply), and the goal is to minimize the total energy consumed in processing all tasks. This scheduling problem can be formulated as a problem of scheduling with rejection, where rejecting a job corresponds to process it on an energy-inefficient machine and has a cost directly proportional to the processing time of the job. To solve this scheduling problem, we introduce a novel $\frac{5}{4}(1+\epsilon)$ approximation algorithm $\mathcal{BEKP}$ by associating it to a Multiple Subset Sum problem. Our algorithm is an improvement over the existing literature, which provides a $(\frac{3}{2} - \frac{1}{2m})$ approximation for scheduling with arbitrary rejection costs. We evaluate and discuss the effectiveness of our approach through a series of experiments, comparing it to existing algorithms.

Keywords: Scheduling with rejection · Approximation algorithm · Energy minimization.

1 Introduction

In this paper, we consider a scheduling problem with the possibility of job rejection, where the cost of rejecting a job is proportional to its processing time. The inspiration for this problem comes from the recent existence of cloud providers that recycle the heat produced by computation to heat buildings or other facilities. For example, Qarnot Computing[4] [3] installs its computing units in water boilers that are used in swimming pools or large apartment buildings to provide hot water. As computing jobs run on these boilers, the energy used for computing is reused to heat the water, eliminating the need for costly air conditioning to cool the machine and reducing the overall carbon footprint.

[4] https://qarnot.com/

However, there may not be enough of these energy-efficient machines available to run all of the required jobs in a reasonable amount of time. In this case, it may be necessary to offload some of this workload to other less efficient compute units, either within the cloud provider or to another conventional public provider. From the point of view of scheduling jobs on the boiler, this corresponds to *rejecting* some jobs, and the goal is to minimize the additional energy required to process these jobs on the less efficient machines. As a first approximation, it is reasonable to assume that the energy required to run a job is proportional to its processing time.

Scheduling with rejection is a valuable concept applicable in various real-life scenarios [12]. For instance, it has been used to model the context of make-to-order production with a limited amount of resources [5], where the cost of rejection corresponds to a lack of income from not producing the item. When modeling our problem, we introduce several additional assumptions to formalize the job scheduling problem in a more interesting and simple way. These assumptions, while extending beyond the initial scope, result in a potentially practical model in other contexts. For example, as minimizing the makespan of the accepted jobs plus the total penalty for rejected jobs is a common objective function in the works for scheduling with rejection, we consider the same objective function. This assumption corresponds for example to the case where a customer rents a boiler to process their jobs and wants to minimize the total energy (or financial cost), which is equal to the total usage of the boiler plus the additional cost of outsourcing some of the jobs to other less efficient machines. Another example is the case where a maintenance operation needs to be scheduled on a boiler. Similarly, to simplify our problem, as a first step, we assume an offline setting and no deadline constraints on the jobs.

The best-known approximation algorithm for scheduling with rejection is reported in a paper by Liu and Lu [8], with an approximation factor of $\frac{3}{2}$. The main contribution of our paper is to exploit the assumption that the rejection cost is proportional to the processing time to obtain a polynomial-time practical approximation algorithm with an improved approximation factor of $\frac{5}{4}(1+\epsilon)$ for any positive ϵ. Our algorithm is based on techniques adapted from an algorithm for the Multiple Subset Sum problem [4], together with ad hoc bounds on the makespan of an optimal solution.

The rest of the paper is organized as follows. We discuss related work and our problem formulation in Sections 2 and 3, respectively. In Section 4, we present a scheduling algorithm for the case where we are given a target makespan T. This algorithm provides a solution with a makespan of at most $\frac{5}{4}T$ with a guarantee on the cost of rejected jobs. In Section 5, we use bounds on the makespan of an optimal solution to approximate it within a factor of $1+\epsilon$ in reasonable time. In Section 6, we perform an experimental study to assess the practical behavior of our algorithm compared to prior solutions. Finally, we present conclusions and perspectives in Section 7.

2 Related Works

For a comprehensive overview of scheduling with rejection, we refer the reader to the surveys by Slotnick [13] and Shabtay et al. [12].

Bartal et al. [1] introduced the problem of scheduling with job rejection. Their objective is to minimize the makespan of accepted jobs plus the sum of penalties associated with rejected jobs. For the online setting, they present a $(1+\phi)$-competitive algorithm, where ϕ represents the golden ratio. For the offline setting, they introduce a fully polynomial approximation algorithm for fixed m and a polynomial approximation algorithm for arbitrary m. In addition, they propose a $(2-\frac{1}{m})$ approximation algorithm for the offline problem with $\mathcal{O}(n \log n)$ complexity.

Ou et al. [11] improve the approximation of Bartal et al. with a heuristic that achieves a worst-case bound of $\frac{3}{2}+\epsilon$ and $\mathcal{O}(n \log n + \frac{n}{\epsilon})$ complexity. Liu and Lu [8] provide a $(\frac{3}{2}-\frac{1}{2m})$ approximation algorithm with $\mathcal{O}(n^3 logn)$ complexity, improving on the work of Ou et al. [11]. In addition to this solution for the identical release date problem, Liu and Lu also present solutions for both the single machine and the parallel machine problems in presence of release dates.

The problem of scheduling with rejection has also been investigated, with the goal of optimizing the sum of weighted completion times of scheduled jobs plus the sum of penalties for rejected jobs. Engels et al. [6] propose general techniques to address offline scheduling with rejection problems with this objective. Epstein et al. [7] focus on the single-machine online problem, where jobs have unit processing times and the weight of each job's completion time is equal to 1. Liu [9] considers the single-machine problem with partial rejection and devises both polynomial-time optimal and pseudopolynomial-time optimal algorithms.

In their work, Mor and Shabtay [10] explore two objectives in scheduling with rejection for single-machine problems. One approach aims to minimize the sum of total late work and rejection cost, while another focuses solely on minimizing total rejection cost, providing an upper bound on total late work.

In our paper, we present a new heuristic to improve on the $(\frac{3}{2}-\frac{1}{2m})$ approximation provided by Liu and Lu [8]. In our setting, the rejection costs are proportional to the processing times of the jobs. We then relate the problem to a Multiple Subset Sum Problem (MSSP).

MSSP consists in allocating a set of n items into m identical bins, each with a positive capacity c. Each item i has a positive weight w_i. The goal is to distribute the items among the bins so as to maximize the total sum of weights of the items in the bins. This problem is known to be strongly NP-hard, and finding an optimal solution is challenging. However, solving MSSP is of practical importance in various domains such as logistics, cutting and packing, where efficient resource allocation is essential to optimize operations.

Caprara et al. [4] propose an algorithm for MSSP that guarantees to obtain at least a fraction $\frac{3}{4}$ of the maximum possible weight sum. The complexity of this approximation algorithm is $\mathcal{O}(m^2+n)$. In their work, they introduce two basic ideas that are relevant in our context. These ideas are explained below.

First, they divide the set of items into five subsets based on their weights relative to the capacity of the bin. In the first step, the subset containing the smallest items is excluded (all other items are considered large). The bounds on the weights of all subsets are used to identify all possible valid *combinations* of large item subsets that can be simultaneously allocated to any bin. This limited set of combinations is then explicitly used to build the approximation algorithm. In the second step, they greedily allocate small items, once other large items have been allocated. They prove that any polynomial-time algorithm that achieves a ratio $\frac{3}{4}$ of the maximum weight sum for MSSP without considering small items can be transformed into one that achieves $\frac{3}{4}$ for the general MSSP (with small items) with the same time complexity. In our work, we adapt these ideas to build the approximation algorithm.

3 Problem Formulation

We consider a scheduling problem where a set of non-preemptive jobs J are to be scheduled on m identical machines. Each job i is characterized by its processing time p_i and can either be processed on one of the energy-efficient machines (corresponding to boilers in the context of Qarnot) or rejected at a cost $\rho \cdot p_i$. This rejection cost represents the cost of offloading the job to other machines (e.g., a public cloud), which are assumed to be in unlimited supply.

A solution $\mathcal{S}$ specifies (i) whether each job is accepted or rejected and (ii) assigns each accepted job i to a machine $j \leq m$. The makespan $C^{\mathcal{S}}$ of a solution is the maximum load on any energy-efficient machine, $C^{\mathcal{S}} = \max_{j \leq m} \sum_{i \text{ assigned to } j} p_i$. We denote as $R^{\mathcal{S}}$ the total processing time (or *area*) of the rejected jobs in $\mathcal{S}$: $R^{\mathcal{S}} = \sum_{i \text{ rejected}} p_i$. The objective of our problem is to minimize the cost $Z^{\mathcal{S}}$, defined as the sum of the occupation of all the machines plus the rejection cost:

$$Z^{\mathcal{S}} = m \cdot C^{\mathcal{S}} + \rho \cdot R^{\mathcal{S}}. \tag{1}$$

Given a target makespan T, we denote with $R^*(T)$ the smallest possible area of rejected jobs among the solutions of makespan at most T. More formally, $R^*(T) = \min_{\mathcal{S}, C^{\mathcal{S}} \leq T} R^{\mathcal{S}}$. This definition leads to the following result:

Lemma 1. *For two values T_1 and T_2 such that $T_1 \leq T_2$, then $R^*(T_2) \leq R^*(T_1)$.*

Table 1 provides a summary of the main notations used in this paper. In the following sections, we present an approximation algorithm for this scheduling problem. We start in Section 4 with finding a good solution when a bound on the makespan is given, and then use this bound in Section 5 to build the overall approximation algorithm.

4 Scheduling with a Bound on Makespan

In this section, we assume that we are given a bound T on the makespan. We present an algorithm called $FillMaxArea$ which, given a set of jobs J, a number of machines m and the bound T, outputs a solution $\mathcal{S}$ with $C^{\mathcal{S}} \leq \frac{5}{4}T$ and $R^{\mathcal{S}} \leq R^*(T)$.

Table 1: Notation employed throughout this paper

m	Number of machines
n	Number of jobs
J	Set of jobs
p_i	Processing time of job i for $i \in \{1, 2, ..., n\}$
W	Area of all jobs in J ($W = \sum_{i \in \{1,2,...,n\}} p_i$)
ρ	Rejection cost coefficient
$C^{\mathcal{S}}$	Makespan of the accepted jobs in schedule $\mathcal{S}$
$A^{\mathcal{S}}$	Area of the accepted jobs in schedule $\mathcal{S}$
$R^{\mathcal{S}}$	Area of the rejected jobs in schedule $\mathcal{S}$
$Z^{\mathcal{S}}$	Cost of the schedule $\mathcal{S}$: $Z^{\mathcal{S}} = mC^{\mathcal{S}} + \rho R^{\mathcal{S}}$
OPT	An optimal schedule which minimizes the cost
$R^*(T)$	Minimum possible area of rejected jobs within makespan T ($min_{\mathcal{S},\, C^{\mathcal{S}} \leq T} R^{\mathcal{S}}$)

4.1 Job Types

From a given makespan bound T, we can group jobs from J according to their processing time. This idea is similar to Caprara et al. [4], using different cutoff values adapted to the context of scheduling with rejection costs.

$$G = \{i \mid \frac{3}{4}T < p_i \leq T\} \qquad N_1 = \{i \mid \frac{1}{2}T < p_i \leq \frac{3}{4}T\}$$
$$N_2 = \{i \mid \frac{3}{8}T < p_i \leq \frac{1}{2}T\} \quad N_3 = \{i \mid \frac{1}{4}T < p_i \leq \frac{3}{8}T\} \quad P = \{i \mid p_i \leq \frac{1}{4}T\}$$

where jobs in G, N_1, N_2, and N_3 are called *long* jobs, while jobs in P are called *short* jobs. We define a *combination* $(Set_1, Set_2, ...)$ as a mapping of jobs to a machine, where exactly one job from each set (a set can occur multiple times) is scheduled on the same machine. For example, (N_3, N_3, N_2) represents two jobs from N_3 and one job from N_2 assigned to a machine in any order.

Lemma 2. *In a schedule with maximum makespan of T, only the following combinations of long jobs are valid:*

$$(G), (N_1), (N_2), (N_3)$$
$$(N_2, N_1), (N_3, N_1), (N_2, N_2), (N_3, N_2), (N_3, N_3)$$
$$(N_3, N_3, N_2), (N_3, N_3, N_3)$$

Proof. Consider one machine in a schedule, with makespan at most T. We split the proof depending on the number l of long jobs this machine processes.

For $l = 1$, any long job guarantees that the makespan bound T is respected. Thus, singleton possibilities are (G), (N_1), (N_2) and (N_3).

For $l = 2$, the combination (N_1, N_1) can not be assigned to that machine: indeed, jobs in N_1 are such that $p_i > \frac{1}{2}T$, so that processing any two of them is not feasible within makespan T. All other combinations of length 2 are valid. Thus, the possible pairs are (N_2, N_1), (N_3, N_1), (N_2, N_2), (N_3, N_2), (N_3, N_3).

For $l = 3$, if two jobs from N_2 are assigned to a machine, even assigning one extra N_3 job is not feasible since the total processing time exceeds $(\frac{3}{8}+\frac{3}{8}+\frac{1}{4})T = T$. Thus, set combinations with longer jobs are not valid either. Therefore, the only possible triplets are (N_3, N_3, N_2) and (N_3, N_3, N_3).

Finally, $l \geq 4$ is not feasible, since any long job has $p_i > \frac{1}{4}T$. □

In addition, it is possible to bound the maximum total processing time of any of these combinations.

Lemma 3. *For any of the combinations provided in Lemma 2, the overall processing time of any valid combination is at most* $\frac{5}{4}T$.

Proof. The proof is trivial, by enumerating all valid combinations and summing the upper bounds of its subset for each element.

4.2 Algorithm

$FillMaxArea$ algorithm is based on these two lemmas. By guaranteeing that the long jobs assigned to each machine obey one of the combinations in Lemma 2, we can guarantee that the resulting solution $\mathcal{S}$ satisfies $C^{\mathcal{S}} \leq \frac{5}{4}T$.

Our long job assignment algorithm is based on the $AssignFrom$ routine, whose pseudocode is given in Algorithm 1. Given the list of combinations and the number l of machines, $AssignFrom$ creates l machine assignments by successively picking the jobs with the largest processing times from the first combination of available jobs. For example, $AssignFrom(\{(N_2, N_1), (N_3, N_1), (N_2, N_2)\}, l)$ selects the largest job from N_2 and the largest job from N_1 until one of them is empty, and then proceeds with the combination (N_3, N_1), and so on.

Algorithm 1 $AssignFrom(combs, l)$

```
Result ← ∅
Remove all combinations from combs where at least one set within the combination is empty
while |Result| ≤ l and combs is not empty do
    Denote by (K_1, K_2, ..., K_k) the first combination in combs
    j_1 ← the largest job from K_1
    j_2 ← the largest remaining job from K_2
    Continue until j_k ← the largest remaining job from K_k
    Result = Result ∪ (j_1, j_2, ..., j_k)
    Remove all combinations from combs where at least one set within the combination is empty
return Result
```

The $FillMaxArea$ algorithm, whose pseudocode is given in Algorithm 2, starts by scheduling the long jobs first, and completes the schedule with a greedy assignment of the short jobs without exceeding the makespan bound $\frac{5}{4}T$. To

decide which long jobs to accept, we set values for l_0, l_1, l_2, and l_3 that represent the number of machines running no long job, one long job, two long jobs, and three long jobs, respectively. For each of these cases, we use the $AssignFrom$ routine with a careful ordering of the combinations identified in Lemma 2. If we run out of jobs in this process, we discard the current solution with quadruplet $j = (l_0, l_1, l_2, l_3)$ and move on to the next possible quadruplet solution. Once an assignment has been computed for all possible quadruplets, the result of $FillMaxArea$ is the one that maximizes the total processing time of all assigned jobs.

Algorithm 2 $FillMaxArea(J, m, T)$

Generate G, N_1, N_2, N_3 and P subsets of J
for each $j = (l_0, l_1, l_2, l_3)$ such that $l_0 + l_1 + l_2 + l_3 = m$ and $l_1 + 2l_2 + 3l_3 \leq n$ **do**
    $X_j \leftarrow \emptyset$
    $X_j \leftarrow X_j \cup AssignFrom(\{(G), (N_1), (N_2), (N_3)\}, l_1)$
    $X_j \leftarrow X_j \cup AssignFrom(\{(N_2, N_1), (N_3, N_1), (N_2, N_2), (N_3, N_2), (N_3, N_3)\}, l_2)$
    $X_j \leftarrow X_j \cup AssignFrom(\{(N_3, N_3, N_2), (N_3, N_3, N_3)\}, l_3)$
    if $l_0 + |X_j| < m$ **then**
        Discard X_j and continue
    Add jobs from P greedily (in any order) to X_j, keeping makespan $\leq \frac{5}{4}T$
$X^* = \{X_j | \max_j A^{X_j}\}$
return X^*

4.3 Proof

We now prove a guarantee on the solution produced by $FillMaxArea$: its makespan is at most $\frac{5}{4}T$, and it rejects not more work (in terms of total processing time) than any solution with makespan at most T.

Lemma 4. *For any T, let $\mathcal{S}$ be the solution obtained by $FillMaxArea(J, m, T)$. Then, $C^{\mathcal{S}} \leq \frac{5}{4}T$ and $R^{\mathcal{S}} \leq R^*(T)$.*

Proof. $C^{\mathcal{S}} \leq \frac{5}{4}T$ is a direct consequence of Lemma 3. We focus on proving $R^{\mathcal{S}} \leq R^*(T)$. Let us denote by $\mathcal{S}_0$ any solution with makespan at most T: we aim to prove that $R^{\mathcal{S}} \leq R^{\mathcal{S}_0}$, or equivalently $A^{\mathcal{S}} \geq A^{\mathcal{S}_0}$.

Lemma 2 defines the list of valid combinations for long jobs in $\mathcal{S}_0$. Let $j = (l_0, l_1, l_2, l_3)$ denote the number of machines with zero, one, two, and three long jobs in $\mathcal{S}_0$, respectively, and consider the solution X_j constructed by $FillMaxArea$ for this particular quadruplet. By construction, $A^{\mathcal{S}} \geq A^{X_j}$. Let us now prove that $A^{X_j} \geq A^{\mathcal{S}_0}$.

Let us consider the small jobs first, and distinguish between two possibilities:

Case 1: At least one small job in P is rejected in X_j. Since a small job is only rejected if it cannot be scheduled to finish before $\frac{5}{4}T$, and since the processing

time of any short job is at most $\frac{1}{4}T$, this ensures that all machines have a workload of at least T. Thus, the total number of accepted jobs satisfies $A^{X_j} \geq m \cdot T \geq A^{\mathcal{S}_0}$, since $\mathcal{S}_0$ has a makespan of at most T.

Case 2: All small jobs are accepted in X_j. In this case we can ignore the small jobs and we will prove that $A^{X_j} \geq A^{\mathcal{S}_0}$ when restricted to long jobs. Indeed, since $\mathcal{S}_0$ cannot accept more small jobs than X_j, this will imply $A^{X_j} \geq A^{\mathcal{S}_0}$ for all jobs.

In the following, we will denote *singleton* a machine that processes a single long job, *pair* a machine that processes two long jobs, and *triplet* a machine that processes three long jobs. Both X_j and $\mathcal{S}_0$ have l_1 singletons, l_2 pairs, and l_3 triplets. In the rest of the proof, we show that $\mathcal{S}_0$ can be transformed into a solution that uses the same number of each *type* of jobs as X_j, without decreasing the total accepted area, where the type of a job refers to the specific long job subset to which it belongs. We will use two possible transformations: *replace*, where an accepted job is exchanged for a rejected job with a longer processing time, and *swap*, where two accepted jobs assigned to different machines are swapped. The first operation increases the total accepted area, while the second does not modify it. Along with the transformations, we will make sure to use only *valid* combinations from the list of Lemma 2.

We start the transformation by considering the l_1 singletons. In X_j, the jobs assigned to these machines are the l_1 longest jobs from J. We build $\mathcal{S}_1$ from $\mathcal{S}_0$ by applying a transformation for each of these longest jobs:

1. If it is rejected in $\mathcal{S}_0$, we *replace* it with the smallest job in a singleton of $\mathcal{S}_0$. This increases the total accepted area of $\mathcal{S}_0$.
2. If it is scheduled either in a pair or triplet in $\mathcal{S}_0$, we *swap* this large job with the smallest job in a singleton of $\mathcal{S}_0$.

The resulting schedule is denoted $\mathcal{S}_1$, and satisfies $(\mathcal{P}_1)$: its singletons process the same set of jobs as the singletons of X_j. In particular, the number of each type of job processed by the singletons is the same.

Based on $(\mathcal{P}_1)$, and given that $FillMaxArea$ schedules as many N_1 jobs as possible in the pairs, the number of N_1 jobs present in a pair is not greater in $\mathcal{S}_1$ than in X_j. If the number of N_1 jobs processed on a pair is greater in X_j, then $\mathcal{S}_1$ must contain more (N_2, N_2), (N_3, N_2), or (N_3, N_3) combinations than C_j, and therefore rejects more N_1 jobs (since N_1 jobs cannot be processed on a triplet). We can *replace* any job in such a combination with a rejected N_1 job until the number of N_1 jobs in pairs is the same as in X_j. This results in either (N_2, N_1) or (N_3, N_1) combinations, both of which are valid. The resulting solution is denoted $\mathcal{S}_2$ and satisfies $(\mathcal{P}_1)$ and $(\mathcal{P}_2)$: it processes the same number of N_1 jobs on pairs as X_j.

X_j cannot use less N_2 jobs than $\mathcal{S}_2$ for the pairs, because $FillMaxArea$ prioritizes N_2 jobs over N_3 jobs. Let us assume that the number of N_2 jobs processed on a pair is greater in X_j than in $\mathcal{S}_2$. Then the missing N_2 jobs in $\mathcal{S}_2$ can either be rejected or scheduled in a (N_3, N_3, N_2) triplet. We can *swap* all N_2 jobs in a (N_3, N_3, N_2) combination with N_3 jobs from (N_3, N_2) or (N_3, N_3).

Here, the possible set combinations we get are either (N_2, N_2) or (N_3, N_2) and (N_3, N_3, N_3), which are all valid. If X_j still uses more N_2 jobs in pairs, then there are rejected N_2 jobs in $\mathcal{S}_2$. We can *replace* one N_3 job from a (N_3, N_2) or (N_3, N_3) combination with each of these rejected N_2 jobs. This results in (N_2, N_2) or (N_3, N_2) valid combinations. The resulting solution is denoted $\mathcal{S}_3$ and satisfies $(\mathcal{P}_1)$, $(\mathcal{P}_2)$ and $(\mathcal{P}_3)$: it processes the same number of N_2 and N_3 jobs on pairs as X_j.

Finally, if X_j schedules more N_2 jobs on triplets than $\mathcal{S}_3$, this implies that there are rejected N_2 jobs in $\mathcal{S}_3$. We can *replace* one N_3 job from a (N_3, N_3, N_3) combination of $\mathcal{S}_3$ with each of these rejected N_2 jobs. This results in a valid (N_3, N_3, N_2) combination. This solution is denoted $\mathcal{S}_4$, and since it satisfies $(\mathcal{P}_1)$, $(\mathcal{P}_2)$, $(\mathcal{P}_3)$ in addition to having the same number of N_2 jobs in triplets, we have shown that $\mathcal{S}_4$ uses the same number of G, N_1, N_2, and N_3 jobs as X_j.

Finally, we use the fact that when choosing a job from a long job set, $FillMaxArea$ always chooses the largest available job. This implies that $A^{X_j} \geq A^{\mathcal{S}_4}$. Since all transformations either increase or do not change the accepted area, we know that $A^{\mathcal{S}_4} \geq A^{\mathcal{S}_0}$, which concludes the proof. □

From this lemma, we can deduce the following bound on the cost of $\mathcal{S}$:

Lemma 5. *For any T, let $\mathcal{S}$ be the solution obtained by $FillMaxArea(J, m, T)$. We can bound its cost by: $Z^{\mathcal{S}} \leq \frac{5}{4}Tm + \rho R^*(T)$.*

Proof. This follows directly from $Z^{\mathcal{S}} = mC^{\mathcal{S}} + \rho R^{\mathcal{S}}$ and Lemma 4. □

5 $\mathcal{BEKP}$ Approximation Algorithm

If we have an optimal solution OPT with respect to the objective function Z, we can compute the solution $FillMaxArea(J, m, C^{OPT})$. From Lemma 5 we get a $\frac{5}{4}$-approximation. In this section, we show how to obtain an approximation of C^{OPT} (with ϵ as the precision coefficient) with controlled complexity. In the end, we obtain the $\mathcal{BEKP}$ algorithm, which is a $\frac{5}{4}(1+\epsilon)$ approximation for any positive number ϵ.

The idea behind $\mathcal{BEKP}$ is to first compute an upper bound U and a lower bound L on the optimal makespan T^{OPT}, and then build different schedules with the $FillMaxArea$ algorithm for each makespan value C_i such that

$$C_i \in \{L, (1+\epsilon)L, ..., (1+\epsilon)^k L\}. \tag{2}$$

The number of iterations k is the smallest value that satisfies $(1+\epsilon)^k L \geq U$, and can be computed as $k = \lceil \log_{1+\epsilon}(\frac{U}{L}) \rceil$. We will now show how to compute U and L so that $\frac{U}{L}$ is bounded, which provides a bound for k.

5.1 Computing Bounds on the Optimal Makespan

Let us define the following function:

$$f(C) = Cm + \rho R^*(C), \tag{3}$$

which represents the minimum possible cost for a schedule with a makespan of C, since $R^*(C)$ is the minimum possible area of rejected jobs for any schedule with that makespan.

We start by providing two lower bounds on $f(C)$. The first one is:

$$f(C) \geq Cm. \tag{4}$$

For the second one, given the total workload $W = \sum_i p_i$, we know that $Cm + R^*(C) \geq W$, which implies that $R^*(C) \geq W - Cm$. Together with (3), this yields the second lower bound:

$$f(C) \geq \rho W - (\rho - 1)Cm. \tag{5}$$

Let us also define a target value H as $H = \dfrac{4\rho W}{5}$.

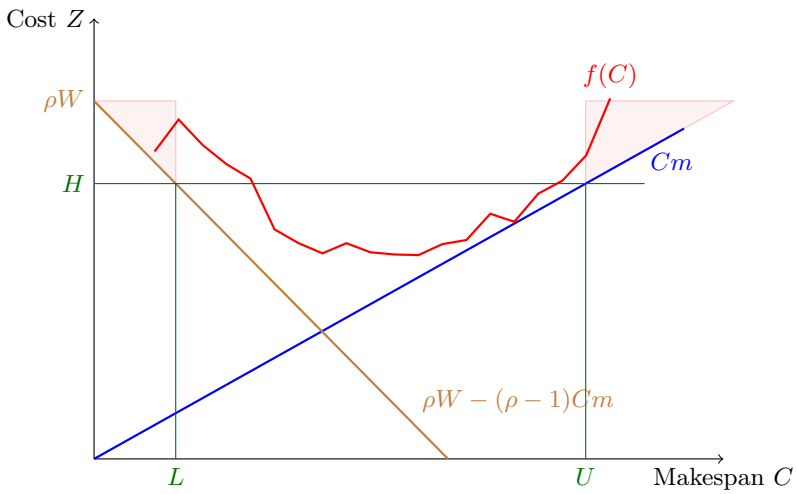

Fig. 1: Sketch of the graph of $f(C)$ (in red), highlighting how the bounds U and L are computed

Let us sketch the possible graph of the cost function $f(C)$ and the two bounds in Fig. 1. The function $f(C)$ is shown in red, the first bound (4) is shown with a blue line, and the second bound (5) is shown in brown. Finally, the target value H is displayed as a green horizontal line.

We define U and L as the values of C such that the first and second bounds are equal to H. This is shown in Fig. 1, and we get $U = \frac{4\rho W}{5m}$. Similarly, given the second bound (5) and the target, we can compute their intersection: $L = \frac{\rho W}{5m(\rho-1)}$.

These values for U and L yield a ratio $\frac{U}{L} = 4(\rho - 1)$, which gives a bound on the number of iterations k. For example, if we assume a rejection cost coefficient

of $\rho \leq 10$, then $4(\rho-1) \leq 36$. If the precision is set to $1+\epsilon = 1.05$, Equation (2) specifies at most 74 different makespan values, which leads to a practical number of iterations. With these values, $\mathcal{BEKP}$ is a $\frac{5}{4} * 1.05 = 1.3125$ approximation algorithm. The tradeoff between number of iterations and performance guarantee can be adjusted when considering different values for ϵ.

5.2 $\mathcal{BEKP}$ Algorithm

Algorithm 3 $\mathcal{BEKP}(J, m)$

1: $X_0 =$ the solution where all jobs are rejected
2: $U = \frac{4\rho W}{5m}$ and $L = \frac{\rho W}{5m(\rho-1)}$ and $k = \lceil \log_{1+\epsilon} \frac{U}{L} \rceil$
3: **for** each $C_i \in \{L, (1+\epsilon)L, (1+\epsilon)^2 L, ..., (1+\epsilon)^k L\}$ **do**
4: $\quad X_i = FillMaxArea(J, m, C_i)$
5: **return** schedule with the lowest cost among X_0 and all X_i

$\mathcal{BEKP}$ is specified in Algorithm 3, where ϵ is a fixed parameter. The algorithm considers several possible schedules: the solution where all jobs are rejected, denoted by X_0, whose cost is $Z^{X_0} = \rho W$, and the result of $FillMaxArea(J, m, C_i)$ for each value C_i between L and U as in Equation (2). The result of $\mathcal{BEKP}$ is the lowest cost schedule among all these candidates.

Theorem 1. *For any positive ϵ, $BEKP$ is a $\frac{5}{4}(1+\epsilon)$ approximation algorithm.*

Proof. Consider an arbitrary set of jobs J to be scheduled on m machines. Let OPT be a schedule for this instance with optimal cost: we will compare Z^{OPT} with the cost of one of the X_i schedules considered in $\mathcal{BEKP}$. We consider two cases, depending on the value of C^{OPT} relative to L and U:

If $C^{OPT} < L$ or $C^{OPT} > U$, we know from $Z^{OPT} \geq f(C^{OPT})$ and our lower bounds (4) and (5) that $Z^{OPT} \geq H = \frac{4\rho W}{5}$. On Fig. 1 this can be interpreted as $f(C^{OPT})$ being located in one of the red triangle areas. Since $Z^{X_0} = \rho W$, we get $Z^{X_0} \leq \frac{5}{4} Z^{OPT}$.

If $L \leq C^{OPT} \leq U$, then there exists an index i such that $C^{OPT} \leq C_i \leq (1+\epsilon)C^{OPT}$. Let us denote this solution as X_i. By Lemma 5 we know that $Z^{X_i} \leq \frac{5}{4} m C_i + \rho R^*(C_i)$. Since $C^{OPT} \leq C_i$, we obtain by Lemma 1 that $R^*(C_i) \leq R^*(C^{OPT})$. Finally, since $C_i \leq (1+\epsilon)C^{OPT}$, we can derive:

$$Z^{X_i} \leq \frac{5}{4}(1+\epsilon) m C^{OPT} + \rho R^*(C^{OPT}) \leq \frac{5}{4}(1+\epsilon) Z^{OPT}$$

In both cases, we identify a schedule X considered by $\mathcal{BEKP}$ that satisfies $Z^X \leq \frac{5}{4}(1+\epsilon) Z^{OPT}$. Since the result of $\mathcal{BEKP}$ has a cost not greater than X, this concludes the proof. □

5.3 Complexity

As discussed in Section 5.1 and the ratio $\frac{U}{L} = 4(\rho - 1)$, the number of calls to $FillMaxArea$ in Algorithm 3 is $\mathcal{O}(\log_{1+\epsilon} \rho)$. In $FillMaxArea$ (Algorithm 2), the number of quadruplets to test is $\mathcal{O}(m^3)$, and for each of them we call $AssignFrom$ and greedily schedule the jobs in P. For a quadruplet, the complexity of all $AssignFrom$ calls is $\mathcal{O}(m)$ in total. We can assume that the jobs are sorted by increasing processing time at the beginning of $\mathcal{BEKP}$, incurring a one-time $\mathcal{O}(n \log n)$ complexity. Scheduling the jobs greedily can be done in $\mathcal{O}(n)$.

In total, the complexity of $\mathcal{BEKP}$ is $\mathcal{O}(m^3(m+n) \log_{1+\epsilon} \rho)$. This can be compared to the algorithm proposed by Liu and Lu [8], whose complexity is $\mathcal{O}(n^3 \log n)$: we expect our approach to be significantly faster in scenarios with fewer machines and a larger number of jobs.

6 Experiments

In this section, we evaluate $\mathcal{BEKP}$ in terms of total solution cost (computed as in Equation 1). To provide reference points, we compare our approach to two existing solutions: a naive solution $\mathcal{LPT}$ that accepts all jobs and schedules them using the Longest Processing Time-first method, and the algorithm proposed by Liu et al. [8], denoted $\mathcal{LIULU}$. Each method has been implemented in a straightforward manner without deep emphasis on performance optimization. In addition, we also compute a lower bound on the solution cost, with an Integer Linear Programming formulation that estimates the makespan of a set of jobs with the standard lower bounds $\max_i p_i$ and $\frac{\sum_i p_i}{m}$, and optimally decides which jobs to accept. This can be formulated as minimizing $Cm + \sum_{i \in J} \rho(1 - x_i)p_i$, subject to the constraints $\forall i \in J, C \geq x_i p_i$ and $C \geq \sum_{i \in J}(x_i p_i)/m$, where x_i is a boolean decision variable equal to 1 if the job is accepted and 0 otherwise. Our simulation code is available as free software in [2]. All experiments were performed sequentially on the Miriel nodes (each consisting of two INTEL Xeon E5-2680v3 12-core 2.50 GHz processors with 128 GB of memory) of the Plafrim supercomputer[5].

We generated random instances in which job processing times follow a lognormal distribution with a mean of 3. We use three different values for the standard deviation σ: 0.5, 0.7, and 1.0. As σ increases from smaller to larger values, the variance in processing times between jobs also increases. We set the number of machines m to 20. We present results for two values of ρ: 1.5 and 4. For each case, we generate 30 different random sets of jobs.

In Fig. 2, each grid column corresponds to a different rejection coefficient, while each row corresponds to a different number of jobs. The x axis represents the different standard deviations used to generate the processing times, and the y axis represents the relative cost of each method compared to the lower bound (where a limit is set for better visualization). The results for each method over

[5] `https://www.plafrim.fr`

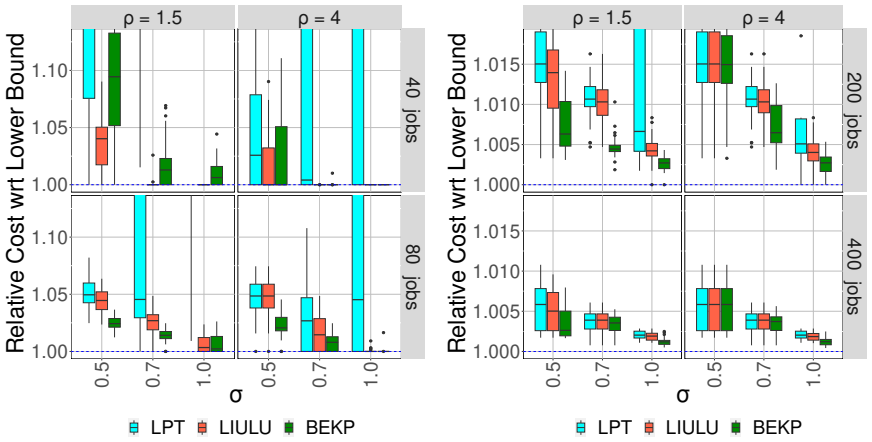

Fig. 2: Comparison of $\mathcal{LIULU}$, $\mathcal{LPT}$ and $\mathcal{BEKP}$ using $m = 20$ for different number of jobs, different values for ρ and σ

the 30 random instances are shown with a boxplot showing the median, first and third quartiles, with whiskers extending to the lowest and highest values. Small black dots represent outliers.

In Fig. 2, we observe that $\mathcal{LPT}$ can yield significantly higher cost solutions compared to the other two methods, especially when there is a small number of jobs and a large variance in the processing times of the jobs. In our experiments, this high cost reached up to 4 times the value of the lower bound. On the other hand, both $\mathcal{BEKP}$ and $\mathcal{LIULU}$ provide low-cost solutions thanks to their rejection capabilities, with neither exceeding the lower bound by more than a factor of 1.2. $\mathcal{BEKP}$ consistently achieves results close to $\mathcal{LIULU}$, and in most cases provides improved solutions at reasonable cost.

7 Conclusion and Perspectives

We address an offline job scheduling problem where jobs are assigned to a limited set of energy-efficient machines, with the option of offloading them to less energy-efficient machines when necessary. This problem can be viewed as a scheduling problem with rejection, where rejection means using less energy-efficient machines, with an energy overhead proportional to the job processing time. We introduce $\mathcal{BEKP}$, a novel $\frac{5}{4}(1+\epsilon)$ approximation algorithm with a time complexity of $\mathcal{O}(m^3(m+n\log n))$. In comparison, the state-of-the-art algorithm of Liu and Lu [8] provides a $(\frac{3}{2}-\frac{1}{2m})$-approximation ratio with a time complexity of $\mathcal{O}(n^3\log n)$ for scheduling tasks with arbitrary rejection costs. Therefore, our proposed algorithm improves both the approximation ratio and the algorithmic complexity with respect to the total number of jobs. Our experimental evaluation shows that our algorithm also produces good quality solutions in practice, in

most cases with similar or better cost compared to Liu and Lu's approach. This work opens up interesting perspectives. Improving the algorithmic complexity of the algorithm could help open it up to more practical cases. The assumption that the rejection cost is proportional to the processing time can be extended to other contexts, such as jobs with quality of service requirements, where this realistic assumption could also lead to improved performance guarantees.

Acknowledgments. Our work is done in the context of the Inria – Qarnot Pulse project: `https://www.inria.fr/en/pulse`.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Bartal, Y., Leonardi, S., Marchetti-Spaccamela, A., Sgall, J., Stougie, L.: Multiprocessor scheduling with rejection. SIAM Journal on Discrete Mathematics **13**(1), 64–78 (2000)
2. Beaumont, O., Eyraud-Dubois, L., Korkmaz, E., Pilla, L.L.: Experimental codes and results for the paper "a $5/4(1+\epsilon)$-approximation algorithm for scheduling with rejection costs proportional to processing times". `https://inria.hal.science/hal-04517532`, accessed: March 25, 2024
3. Bouzel, R., Ngoko, Y., Benoit, P., Sainthérant, N.: Distributed grid computing manager covering waste heat reuse constraints. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 294–299. IEEE (2021)
4. Caprara, A., Kellerer, H., Pferschy, U.: A 3/4-approximation algorithm for multiple subset sum. Journal of Heuristics **9**(2), 99–111 (03 2003)
5. Cesaret, B., Oğuz, C., Sibel Salman, F.: A tabu search algorithm for order acceptance and scheduling. Computers & Operations Research **39**(6), 1197–1205 (2012), special Issue on Scheduling in Manufacturing Systems
6. Engels, D.W., Karger, D.R., Kolliopoulos, S.G., Sengupta, S., Uma, R.N., Wein, J.: Techniques for scheduling with rejection. In: Bilardi, G., Italiano, G.F., Pietracaprina, A., Pucci, G. (eds.) Algorithms — ESA' 98. pp. 490–501. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
7. Epstein, L., Noga, J., Woeginger, G.J.: On-line scheduling of unit time jobs with rejection: minimizing the total completion time. Operations Research Letters **30**(6), 415–420 (2002)
8. Liu, P., Lu, X.: New approximation algorithms for machine scheduling with rejection on single and parallel machine. Journal of Combinatorial Optimization **40**(4), 929–952 (2020)
9. Liu, Z.: Scheduling with partial rejection. Operations Research Letters **48**(4), 524–529 (2020)
10. Mor, B., Shabtay, D.: Single-machine scheduling with total late work and job rejection. Computers & Industrial Engineering **169**, 108168 (2022)
11. Ou, J., Zhong, X., Wang, G.: An improved heuristic for parallel machine scheduling with rejection. European Journal of Operational Research **241**(3), 653–661 (2015)
12. Shabtay, D., Gaspar, N., Kaspi, M.: A survey on offline scheduling with rejection. Journal of scheduling **16**, 3–28 (2013)
13. Slotnick, S.A.: Order acceptance and scheduling: A taxonomy and review. European Journal of Operational Research **212**(1), 1–11 (2011)