A Robust PTAS for Machine Covering and Packing*

Martin Skutella and José Verschae

Institute of Mathematics, TU Berlin, Germany,
{skutella,verschae}@math.tu-berlin.de

Abstract. Scheduling a set of n jobs on m identical parallel machines so as to minimize the makespan or maximize the minimum machine load are two of the most important and fundamental scheduling problems studied in the literature. We consider the general online scenario where jobs are consecutively added to and/or deleted from an instance. The goal is to always maintain a (close to) optimal assignment of the current set of jobs to the m machines. This goal is essentially doomed to failure unless, upon arrival or departure of a job, we allow to reassign some other jobs. Considering that the reassignment of a job induces a cost proportional to its size, the total cost for reassigning jobs must preferably be bounded by a constant r times the total size of added or deleted jobs.
Our main result is that, for any $\varepsilon > 0$, one can always maintain a $(1+\varepsilon)$-competitive solution for some constant reassignment factor $r(\varepsilon)$. For the minimum makespan problem this is the first improvement of the $(2+\varepsilon)$-competitive algorithm with constant reassignment factor published in 1996 by Andrews, Goemans, and Zhang. The crucial factor for obtaining this result is a novel insight into the structure of robust, almost optimal schedules. Here, the term *robust* refers to the fact that, upon arrival or departure of a job, reoptimization requires only slight changes of the current schedule.

KEY WORDS: scheduling, approximation, online, reassignment costs, robustness, ILP in constant dimension

1 Introduction

We consider two basic scheduling problems where n jobs need to be assigned to m identical parallel machines. Each job j has a non-negative processing time p_j and the load of a machine is the total processing time of jobs assigned to it. The *machine covering problem* asks for an assignment of jobs to machines that maximizes the minimum machine load. In the *minimum makespan problem* (or *machine packing problem*), we wish to find a schedule minimizing the maximum machine load.

Both problems are well known to be strongly NP-hard and both allow for a polynomial-time approximation scheme (PTAS); see, e. g., [2,9,17]. They have

* This work was partially supported by Berlin Mathematical School and by DFG research center MATHEON in Berlin.

also been studied extensively in the online setting where jobs arrive one by one and must immediately be assigned to a machine at their arrival; see, e. g., [1,15]. The best known online algorithm for the minimum makespan problem is a 1.9201-competitive algorithm due to Fleischer and Wahl [8]. The best lower bound 1.88 on the competitive ratio of any deterministic online algorithm currently known is due to Rudin and Chandrasekaran [11]. For randomized online algorithms there is a lower bound of $e/(e-1) \approx 1.58$; see Chen, Vliet, and Woeginger [6] and Sgall [14].

The online variant of the machine covering problem turns out to be less tractable and there is no online algorithm with constant competitive ratio. The best possible deterministic algorithm greedily assigns jobs to the least loaded machine, and has competitive ratio m; see Woeginger [17]. Azar and Epstein [4] show a lower bound of $\Omega(\sqrt{m})$ for the competitive ratio of any randomized online algorithm, and give an almost matching $\tilde{O}(\sqrt{m})$-competitive algorithm.

Proportional reassignment cost. We study a relaxed online scenario known as *online load balancing with proportional reassignment cost*. In this setting, jobs may arrive or depart at any time, and when a new job enters the system it must immediately be assigned to a machine. Again, the objective is either to minimize the makespan or to maximize the minimum machine load. Furthermore, upon arrival or departure of a job, one is allowed to reassign other jobs by paying an associated cost: reassigning job j incurs a cost of $c \cdot p_j$ for some given constant $c > 0$. By scaling we can assume that $c = 1$.

The cost due to reassignments is controlled by the *reassignment factor* which is defined as follows. Let J be the set of jobs that have so far appeared in the system, and let $J_L \subseteq J$ be the set of jobs that have left the system. We define the reassignment factor r of an algorithm as the worst case ratio between $\sum_{j \in J} p_j + \sum_{j \in J_L} p_j$ and the total cost due to reassignments. Alternatively, we can interpret this framework in the following way: given a parameter $r > 0$, the arrival or departure of a job j adds an amount of $r \cdot p_j$ to the total budget available to spend on reassignments. We call $r \cdot p_j$ the *reassignment potential* induced by job j.

Note that $r = 0$ means that no reassignment is allowed, and thus we are in the classical online setting. On the other hand, $r = \infty$ implies that we are allowed to reassign all jobs at each arrival/departure, and thus we fall back to the offline case. We are interested in developing α-competitive algorithms where the migration factor r is bounded by a constant. Furthermore, we study the trade-off between α and r. Arguably, the best that we can hope for under this framework is a *robust PTAS* (also known as *dynamic PTAS*), that is, a family of polynomial-

time $(1+\varepsilon)$-competitive algorithms with constant reassignment factor $r = r(\varepsilon)$, for all $\varepsilon > 0$.

For the minimum makespan problem with proportional reassignment cost, Westbrook [16] gives a 6-competitive algorithm with reassignment factor 1 (according to our definition[1]). Andrews, Goemans, and Zhang [3] improve upon this result, obtaining a 3.5981-competitive algorithm with reassignment factor 1. Furthermore, they give a $(2 + \varepsilon)$-competitive algorithm with constant reassignment factor $r(\varepsilon) \in O(1/\varepsilon)$.

Related work. Sanders, Sivadasan, and Skutella [12] consider a somewhat tighter online model, known as the *bounded migration* framework. This model can be interpreted as the reassignment model with the following modification: after the arrival or departure of a job j, its reassignment potential $r \cdot p_j$ must be immediately spent or is otherwise lost. In the bounded migration scenario, the value r is called the *migration factor* of the algorithm, and is a measure of the robustness of the constructed solutions.

Sanders et al. study the bounded migration model for the special case when jobs are not allowed to depart. For the minimum makespan problem, they give a $3/2$-competitive algorithm with migration factor $4/3$. Moreover, using well known rounding techniques, they formulate the problem as an integer linear programming (ILP) feasibility problem in constant dimension. Combining this with an ILP sensitivity analysis result, they obtain a robust PTAS for the bounded migration model with job arrivals only. An important consequence of their analysis is that no special structure of the solutions is needed to achieve robustness. More precisely, it is possible to take an arbitrary $(1 + \varepsilon)$-approximate solution and, at the arrival of a new job, turn it into a $(1+\varepsilon)$-approximate solution to the augmented instance while keeping the migration factor constant. This feature prevents their technique from working in the job departure case.

Sanders et al. [12] also consider the machine covering problem, showing a 2-competitive algorithm with migration factor 1. Moreover, they give a counterexample showing that it is not possible to start with an arbitrary $(1/2 + \varepsilon)$-approximate solution, and then maintain the approximation guarantee while keeping the migration factor constant. This implies that the ideas developed in [12] for the minimum makespan problem cannot be applied directly to derive a robust PTAS for the machine covering problem. Based on ideas in [12], Epstein and Levin [7] develop a robust APTAS for the Bin-Packing problem.

[1] Our definition differs slightly from the one given in [16]: they do not consider the departure of jobs to add any reassignment potential, and the first assignment of a job also induces cost in their case. However, the concept of constant reassignment factors is the same in both models.

The work of Sanders et al. is motivated by an important practical online application. A Storage Area Network (SAN) commonly manages several disks of different capacities. For the system to be fault-tolerant, the information is usually replicated several times. We can model a SAN by considering a partition of the storage devices into several subservers, each containing copies of the same information. Therefore, the capacity of the SAN is the minimum capacity of all subservers. In scheduling notation, the subservers corresponds to machines, while the disks correspond to jobs. Our objective is to maximize the capacity of the SAN, i.e., the minimum load of the machines. Moreover, we might want to increase the capacity of the SAN by attaching new disks. This corresponds to jobs that enter the system in our online model. On the other hand, job departure models disks that fail and must be removed from the network. We would like to maintain solutions that are close to optimal, by reassigning a limited amount of hard disks. However, the more hard disks that arrive or fail, the more reassignments we are be willing to perform. Notice that this problem fits the reassignment and the bounded migration models. Nonetheless, the latter unrealistically asks the reassignment potential to be spent immediately after is generated. This may be undesirable in practice since it may provoke down-time of the system each time a new disk is inserted, instead of collecting work until it is worthy to make a larger change of configuration.

Our Contribution. Using several novel ideas and techniques, we develop a general framework for obtaining robust PTASes in the reassignment model. Our results can be considered from various different angles and have interesting interpretations in several different contexts:

(i) We make a significant contribution to the understanding of two fundamental *online scheduling* problems on identical parallel machines that are also relevant building blocks for many more complex real-world problems.
(ii) We advance the understanding of *robustness* of parallel machine schedules under job arrival and departure, and give valuable insights related to the *sensitivity analysis* of parallel machine schedules.
(iii) We achieve the best possible performance bound for *machine balancing* with proportional reassignment costs, improving upon earlier work by Westbrook [16] and Andrews, Goemans, and Zhang [3].

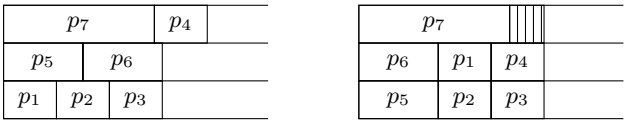
Our techniques for deriving the robust PTAS take the ideas in [2] and [12] one step further. We first prove that it is not possible to start with an arbitrary $(1+\varepsilon)$-approximate solution and, at the arrival of a new job, maintain the competitive ratio with constant migration factor. One of our main contributions is to overcome this limitation by giving extra structure to the constructed solutions.

Roughly speaking, we do this by asking for solutions such that the sorted vector of machine load values is lexicographically optimal. It turns out that a solution with this property is not only optimal but also robust. In the analysis we formulate a rounded scheduling problem as an ILP in constant dimension, exploit the structure of the coefficient matrix, and apply sensitivity analysis for ILPs to derive the result. This, plus other techniques, allows us to obtain a robust PTAS with constant reassignment factor for the case where jobs are allowed to arrive and depart.

Our techniques can be further improved and extended. Our robust PTAS can be refined so that we only accumulate a small amount of reassignment potential, being at most $\varepsilon \cdot OPT$. Note that if we do not accumulate any reassignment potential, we fall back to the bounded migration case. This implies that our techniques are best possible, since we will show that there is no robust PTAS with bounded migration. On the other hand, all our techniques can be extended to a very broad class of problems, first consider by Alon, Azar, Woeginger and Yadid [2], where the objective functions solely depend on the load of each machine.

Organization of the paper. To keep the presentation short and clear, we mainly focus on the machine covering problem and present a robust PTAS for the general case of jobs leaving and entering the system. In the end we show how to obtain this result for the minimum makespan problem and other objective functions.

In Section 2 we give a lower bound on the best possible competitive guarantee that we can obtain in the constant migration model. We also discuss how to deal with small arriving/departing jobs. Section 3 shows how to compute a lower bound on the minimum load that is stable against arrival or departure of jobs. Section 4 is devoted to show our main structural insights. There we describe, for a particular case, an important property that guarantees robustness of solutions. Afterwards, in Section 5, we generalize these ideas to derive a robust PTAS with constant reassignment factor for the machine covering problem. In Section 6 we refine our techniques so that our algorithm does not accumulate more than $\varepsilon \cdot OPT$ reassignment potential. Finally, Section 7 explains how our techniques can be applied to derive robust PTASes for a broad class of objective functions. In particular, we show this result for the minimum makespan problem, improving upon the $(2+\varepsilon)$-competitive algorithm with constant reassignment factor by Andrews, Goemans, and Zhang [3]



(a) Unique optimal solution to original instance.

(b) Unique optimal solution to instance with new jobs.

Fig. 1: There is no $(19/20 + \varepsilon)$-competitive algorithm with constant migration factor.

2 A lower bound on the best approximation with constant migration factor

We start by showing that it is not possible to maintain near-optimal solutions to the machine covering problem with constant migration factor in the model of Sanders et al. [12], if arriving jobs are arbitrarily small.

Lemma 1. *For any $\varepsilon > 0$, there is no $(19/20+\varepsilon)$-competitive algorithm for the machine covering problem with constant migration factor, even for the special case without job departures.*

Proof. Consider an instance consisting of 3 machines and 7 jobs of sizes $p_1 = p_2 = p_3 = p_4 = 2$, $p_5 = p_6 = 3$ and $p_7 = 5.7$. It is easy to see that the optimal solution is given by Figure 1a. Moreover, this is, up to symmetry, the only solution within a factor $19/20 + \varepsilon$ to the optimum for any $\varepsilon > 0$.

Let us assume by contradiction that we can obtain a $(19/20+\varepsilon)$-competitive algorithm with constant migration factor C. Then, we must start with the solution given by Figure 1a. Consider now that jobs of size less than $1/C$ arrive, whose total processing time sum up to 1.3. Since the migration factor is at most C, non of the seven original jobs can change machines, and thus the best possible solution we can achieve has value 6.65, while the optimum, shown in Figure 1b, is 7. We conclude by noting that $6.65/7 = 19/20$. □

As mentioned before, this lemma justifies the use of the reassignment cost model instead of the bounded migration framework. Moreover, we see in the proof of the lemma that the limitation of the bounded migration model is caused by arbitrarily small jobs, whose reassignment potential do not allow any other job to be migrated. Nonetheless, in the reassignment model we can deal with small jobs by accumulating them as follows.

Let OPT denote the value of an optimum solution for the current set of jobs. If a new job j with $p_j \leq \varepsilon \cdot OPT$ arrives, we do not schedule it immediately[2]. Instead, we accumulate several small jobs, until their total processing time surpasses $\varepsilon \cdot OPT$. We can then incorporate them as one larger job with processing time at least $\varepsilon \cdot OPT$. This can only decrease the value of the solution by a $1-\varepsilon$ factor.

The situation for the departure of small jobs is slightly more complicated. We ignore the fact that certain small jobs are gone as long as the following property holds: There is no machine which has lost jobs of total processing time at least $\varepsilon \cdot OPT$. Under this condition, the objective function is affected by less than a factor $1-\varepsilon$. If, on the other hand, there is such a machine, we can treat the set of jobs that have left the machine as one single job of size at least $\varepsilon \cdot OPT$ and act accordingly. Notice that the property above has to be checked dynamically after each reassignment of jobs caused by newly arriving or departing jobs.

Assumption 1 *We assume, without loss of generality, that all arriving/departing jobs are larger than $\varepsilon \cdot OPT$.*

3 A stable estimate of the optimum value

In this section we show how to compute an upperbound on the machine covering problem, that is within a factor 2 of the optimum. We use here the same upperbound as in [2]. Nonetheless, for it to be useful for the robust PTAS, we need this upperbound to be stable: at the arrival/departure of a new job, its value must not change by more than a constant factor. We first describe the lower bound when the instance satisfies a special property. We show later how to generalize this to arbitrary instances.

Let $\mathcal{I} = (J, M)$ be an instance of our problem, where J is a set of n jobs and M a set of m machines. Given a subset of jobs L, we denote by $p(L)$ the total processing time of jobs in L, i. e., $p(L) := \sum_{j \in L} p_j$.

The most natural upperbound to use for our problem is just the average load of the instance, $p(J)/m$. However, this estimate is clearly not within a constant factor of the optimum (consider, e. g., an instance with two machines and two jobs with processing times 1 and $K \gg 1$, respectively). Throughout this section we say that instance $\mathcal{I}$ satisfies property $(*)$ if $p_j \leq p(J)/m$, for all $j \in J$. Under condition $(*)$, the average load is always within a factor 2 of OPT; see [2]. For completness, we also give the proof of this of this fact.

[2] In order to still satisfy the strict requirements of the considered online scheduling problem, we can assume that job j is temporarily assigned to an arbitrary machine, say machine 1. Notice that this causes an increase of the reassignment factor by at most 1.

Lemma 2. *If instance $\mathcal{I}$ satisfies $(*)$, then $\frac{p(J)}{2m} \leq OPT \leq \frac{p(J)}{m}$.*

Proof. The upperbound is clear. Assume by contradiction that $OPT < p(J)/2m$, and consider an optimal solution minimizing the number of jobs whose starting time is larger than OPT.

Our contradiction hypothesis implies that there must exist some machine i whose load is strictly larger than the average load $p(J)/m$. Since the processing time of every job is at most $p(J)/m$, machine i must contain at least two jobs, $j \neq j'$. We assume without loss of generality that $p_j \leq p_{j'}$, and that j is the last job processed on i.

Note that by our choice of optimal solutions, the starting time of all jobs must be smaller or equal than OPT. Thus, if S_j and C_j denote the starting and completion time of job j respectively, we get that

$$S_j \leq OPT < \frac{p(J)}{m} < C_j \, .$$

We conclude that $p_j = C_j - S_j > p(J)/m - OPT > p(J)/2m$, where the last inequality follows from our contradiction hypothesis. Thus, $p_{j'} \geq p_j > \frac{p(J)}{2m}$, and therefore $S_j > p_{j'} \geq \frac{p(J)}{2m} > OPT$. Hence, moving job j to the machine with minimum load yields a schedule that has one job less whose starting time is larger than OPT. This contradicts out initial assumption on the considered optimal solution. □

Now we show how to transform arbitrary instances to instances satisfying $(*)$ without changing the optimal solution value. If $p_j > p(J)/m \geq OPT$, then we can assume that j is being processed on a machine of its own. Thus, removing j plus its corresponding machine does not change the optimal solution value, but it does reduce the average load. Iterating this idea we get the following simple algorithm.

STABLE-AVERAGE

1. Initialize $w \leftarrow m$ and $L \leftarrow J$.
2. Order the processing times so that $p_{j_1} \geq p_{j_2} \geq \ldots \geq p_{j_n}$.
3. For each $k = 1, \ldots, n$, define

$$A \leftarrow \frac{p(L)}{w},$$

and check whether $p_{j_k} \leq A$. If this holds, then return A together with w and L. Otherwise, redefine $w \leftarrow (w-1)$ and $L \leftarrow (L \setminus \{p_{j_k}\})$ and keep iterating.

We call value A the *stable average* of instance $\mathcal{I}$. Also, we obtain that solving the instance with job set L and w identical machines is equivalent to solving $\mathcal{I}$. This, together with Lemma 2, implies the following.

Lemma 3. *The upperbound A computed above satisfies $OPT \leq A \leq 2 \cdot OPT$.*

It is easy to see that, in general, the factor by which the upperbound changes at the arrival/departure of a job is not bounded (consider two machines and two jobs of sizes 1 and $K \gg 1$, respectively; then one job of size $K - 1$ arrives). However, we can show that if A is increased by more than a factor 2, then the instance was trivial to solve in the first place. We first show that, if the value A is increased by more than a factor of 2, then a significant amount of jobs must have arrived to the system.

Lemma 4. *Consider an arbitrary instance $\mathcal{I}' = (J', M)$, and let A', L' and w' be the returned values when applying Algorithm* STABLE-AVERAGE *to it. If $A' > 2A$, then $|J \triangle J'| > w/2$ (*here $\triangle$ denotes the symmetric difference between the two sets*).*

Proof. Let δ be an arbitrary positive number. We assume, without loss of generality, that jobs in instances $\mathcal{I}$ and $\mathcal{I}'$ have processing times bounded by $A'(1+\delta)$. Indeed, if there is some job j with $p_j > A'$, reducing its processing time to $A'(1 + \delta)$, for any $\delta > 0$, leaves the values of A, A', and w unchanged. Let $k := |J' \setminus J| \leq |J' \triangle J|$, then

$$A' \leq \frac{wA + (m - w)A'(1 + \delta) + kA'(1 + \delta)}{m} .$$

Simple algebraic manipulation yields that

$$k \geq \frac{w\left(1 - \frac{A}{A'}\right) + \delta(m + w)}{(1 + \delta)} .$$

Notice that the limit of the right hand side when $\delta \to 0^+$ equals $w(1 - A/A') > w/2$. The result then follows by choosing δ small enough. □

Moreover, we say that an instance is *trivial* if Algorithm STABLE-AVERAGE returns $w = 1$. If this is the case, then the optimal solution to the instance can be constructed by processing the $m-1$ largest jobs each on a machine of their own, and the remaining jobs on the remaining machine. Moreover, the optimal value OPT equals A. With this definition, we obtain the following easy consequence of Lemma 4.

Corollary 1. *Assume that $\mathcal{I}$ is nontrivial and that instance $\mathcal{I}'$ is obtained from $\mathcal{I}$ by adding one job. Then, it must hold that $A \leq A' \leq 2 \cdot A$.*

4 The structure of robust solutions

In the following, we show a sufficient condition to guarantee that we can achieve near optimal solutions when jobs arrive or depart. For clarity, we first consider a static case: Given an instance $\mathcal{I}$, we construct a $(1-O(\varepsilon))$-approximate solution having enough structure so that at the arrival or departure of a job larger than $\varepsilon \cdot OPT$, we can maintain the approximation guarantee using constant migration factor. Note that since we are using constant migration factor, we only use the reassignment potential induced by the arriving or departing job. Nonetheless, we do not take care of maintaining the structure so that this procedure can be iterated when further jobs arrive (or depart). We deal with this more complicated scenario in Section 5.

In the remainder of this section we concentrate on the case of a newly arriving job. The presented ideas and techniques can be easily adapted to the case of a departing job. Let $\mathcal{I} = (J, M)$ be an arbitrary instance with optimal value OPT. If there is no possible confusion, we will also use OPT to refer to some optimal schedule for $\mathcal{I}$. We call $\mathcal{I}' = (J', M)$ the instance with the additional arriving job p_{j^*}, and OPT' the new optimal value.

The case that $\mathcal{I}$ is trivial is treated separately. As discussed before, in this case we can easily compute an optimal solution to $\mathcal{I}$.

Lemma 5. *Assume that $\mathcal{I}$ is trivial. Then, starting from an optimal solution, it is possible to construct a $(1-\varepsilon)$-approximate solution to $\mathcal{I}'$ by using migration factor at most $2/\varepsilon$.*

Proof. Construct an arbitrary $(1-\varepsilon)$-approximate solution S' to $\mathcal{I}'$, by using for example, the procedure that will be derived in Section 4.1, or the PTASes in [2,17]. Recall that by applying Algorithm STABLE-AVERAGE to instance $\mathcal{I}$, we obtain its stable average A, as well as a subset of jobs L and a number of machines w so that $A = p(L)/w$. Moreover, all jobs $j \in L$ satisfy $p_j \leq A$. We can assume that in solution S' no two jobs in $J' \setminus L$ can be processed on the same machine. Indeed, consider two jobs $j, j' \in J' \setminus L$. Since $|J' \setminus L| = m$, there must be a machine processing only jobs in L. Since $p(L) = A$, interchanging all jobs in L with j cannot decrease the minimum load. Iterating this argument we obtain a schedule S' where all jobs in $J' \setminus L$ are processed on different machines.

With this, up to permutation of machines, schedules S and S' differ only with respect to the new job j^* and jobs in L. Since we are assuming that $p_{j*} \geq \varepsilon \cdot OPT \geq \varepsilon \cdot A/2$, and we have that $p(L) = A$, we conclude that the migration factor needed is at most $2/\varepsilon$. □

In the rest of this section we take care of the nontrivial case.

4.1 Compact description of a schedule

As usual in PTASes, we first simplify our instance by rounding. In this section we briefly show how to do this for our problem. The techniques are similar to the ones found, e. g., in [2,12,17]. Nonetheless, we must be careful to ensure that the resulting compact description of schedules is also compatible with schedules containing any new job that may arrive.

It is a well known fact that by only loosing a $1/(1+\varepsilon)$ factor in the objective function, we can round down all processing times to the nearest power of $1+\varepsilon$. Thus, in the rest of this paper we will assume that, for every job j, it holds that $p_j = (1+\varepsilon)^k$ for some $k \in \mathbb{Z}$. Moreover, we need to compute an upperbound, UB, which is within a constant factor $\gamma > 1$ of the optimal value:

$$OPT \leq \text{UB} \leq \gamma \cdot OPT\,.$$

Throughout this section we use $\text{UB} = A$, so that $\gamma = 2$. For the general case in Section 5 we will have to choose this upperbound more carefully. In what follows, we will round our instance such that the number of different processing times is constant. To this end, let $\sigma, \Sigma \geq 1$ be two constant parameters that will be chosen appropriately later. Our rounding will ensure that all processing times belong to the interval $[\varepsilon \cdot \text{UB}/\sigma, \Sigma \cdot \text{UB}]$. The value σ will be chosen big enough so that every job that is smaller than $\varepsilon \cdot \text{UB}/\sigma$ is also smaller than $\varepsilon \cdot OPT$. On the other hand, since $\Sigma \geq 1$, every job that is larger than $\Sigma \cdot \text{UB}$ will also be larger than OPT, and thus will be processed on a machine of its own. Moreover, since we are assuming that $\mathcal{I}$ is non-trivial, Corollary 1 implies that $\text{UB}' \leq 2 \cdot \text{UB}$. We can therefore choose $\Sigma \geq 2$ to ensure that a job that is larger than $\Sigma \cdot \text{UB}$ is also larger than OPT', and thus will also be processed on a machine of its own in optimal solutions to $\mathcal{I}'$. This will help us to simultaneously round $\mathcal{I}$ and $\mathcal{I}'$, having the same approximation guarantee for both instances. More importantly, we note that since the lower and upper bounds are within a constant factor, the rounded instances only have $O(\log_{1+\varepsilon}(1/\varepsilon)) = O(1/\varepsilon \log(1/\varepsilon))$ different job sizes.

Consider the index set

$$I(\text{UB}) := \left\{ i \in \mathbb{Z} : \varepsilon \cdot \text{UB}/\sigma \leq (1+\varepsilon)^i \leq \Sigma \cdot \text{UB} \right\} = \{\ell, \ldots, u\}\,.$$

The new rounded instance derived from $\mathcal{I}$ is described by defining a vector $N = (n_i)_{i \in I}$, whose entry n_i denotes the number of jobs of size $(1+\varepsilon)^i$. More precisely, vector N is defined as follows. For each $i = \ell + 1, \ldots, u-1$, we let

$$n_i := \left| \left\{ j \in \{1, \ldots, n-1\} : p_j = (1+\varepsilon)^i \right\} \right|, \tag{1}$$

i. e., n_i is the number of jobs of size $(1+\varepsilon)^i$ in the original instance, and thus these jobs are not rounded. From now on we will call such jobs *big* with respect to UB. To get rid of jobs that are even larger, we just round them down to $(1+\varepsilon)^u$ and define

$$n_u := |\{j \in \{1,\dots,n-1\} : p_j \geq (1+\varepsilon)^u\}| \,. \tag{2}$$

We call these jobs *huge* with respect to UB. Finally, jobs that are smaller than or equal to $(1+\varepsilon)^\ell$ are said to be *small* with respect to UB. To dispose of these jobs, we replace them by jobs of size $(1+\varepsilon)^\ell$ by defining

$$n_\ell := \left\lfloor \frac{\sum_{j:p_j \leq (1+\varepsilon)^\ell} p_j}{(1+\varepsilon)^\ell} \right\rfloor . \tag{3}$$

Notice that with definition (3) we make sure that the total processing time of small jobs in N and $\mathcal{I}$ is roughly equal. By slightly abusing notation, in what follows we also use the symbol N to refer to the scheduling instance defined by the vector N.

Lemma 6. *The value of an optimal solution to N is within a $1 - O(\varepsilon)$ factor of OPT.*

Proof. Let us consider an optimal schedule S for $\mathcal{I}$ with objective value OPT. We modify this solution to construct a schedule for N. First, replace all jobs with processing times larger than $(1+\varepsilon)^u$ with a job of size $(1+\varepsilon)^u$. By taking $\Sigma \geq 1+\varepsilon$, the load of each affected machine is still at least $(1+\varepsilon)^u \geq \Sigma \cdot \text{UB}/(1+\varepsilon) \geq OPT$. Then, remove all jobs j with $p_j \leq (1+\varepsilon)^\ell$, and apply a *list scheduling* algorithm to the n_ℓ jobs with size $(1+\varepsilon)^\ell$ of instance N, i. e., greedily assign each job to the least loaded machine in an arbitrary order.

Call j the last job scheduled doing this procedure, and let S_j be its starting time. It is clear that the value of the schedule is at least S_j. Assume by contradiction that $S_j < OPT - 2(1+\varepsilon)^\ell$. Since we are using a greedy algorithm, all jobs of size $(1+\varepsilon)^\ell$ have completion time strictly smaller than $OPT - (1+\varepsilon)^\ell$. Also note that, by the definition of n_ℓ,

$$\sum_{j:p_j \leq (1+\varepsilon)^\ell} p_j - n_\ell (1+\varepsilon)^\ell \leq (1+\varepsilon)^\ell \,.$$

This yields a contradiction since small jobs in $\mathcal{I}$ can be used to cover all machines up to OPT. We conclude that the value of the constructed schedule is at least

$$\begin{aligned} OPT - 2(1+\varepsilon)^\ell &\geq OPT - 2\text{UB}\frac{\varepsilon(1+\varepsilon)}{\sigma} \\ &\geq OPT - 2OPT\frac{\gamma\varepsilon(1+\varepsilon)}{\sigma} = (1-O(\varepsilon))OPT \,, \end{aligned}$$

where the last computations follow from the definition of UB and ℓ. □

Notice that a solution to the rounded instance can be turned into a schedule for the original instance $\mathcal{I}$ by simply removing all jobs of size $(1+\varepsilon)^\ell$ from the schedule of N, and then applying a list scheduling algorithm to process the original small jobs. By the same argument as in the proof of Lemma 6, we can conclude that doing this only costs a factor $1 - O(\varepsilon)$ in the objective function. We can thus restrict to work with instance N, whose jobs take only a constant number of different sizes. To describe a schedule of N in a compact way, we consider the following definition.

Definition 1 (Machine configuration and its load). *For a given schedule, a machine is said to obey configuration $k : I(\text{UB}) \to \mathbb{N}_0$, if $k(i)$ equals the number of jobs of size $(1+\varepsilon)^i$ being processed on that machine, for all $i \in I(\text{UB})$. Also, the load of a configuration k, denoted as $\text{load}(k)$, is the load of a machine that obeys that configuration, i. e., $\text{load}(k) = \sum_{i \in I(\text{UB})} k(i)(1+\varepsilon)^i$.*

Let us now consider the set of configurations

$$K := \{k : I(\text{UB}) \to \mathbb{N}_0 | k(i) \leq \sigma\Sigma/\varepsilon + 1 \text{ for all } i \in I\} \ .$$

In the next lemma we show that these are all necessary configurations that we need to consider, and thus we can restrict to only a constant number of configurations: $|K| \leq (\sigma\Sigma/\varepsilon + 1)^{|I(\text{UB})|} \in 2^{O(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})}$.

Lemma 7. *There exists an optimal solution to N, such that all machines obey a configuration in K.*

Proof. It is enough to show that there can be at most $\sigma\Sigma/\varepsilon+1$ jobs processed on any machine in an optimal schedule. It is easy to see that there exists an optimum solution where no job starts later than UB. Therefore, since all jobs are larger than $\varepsilon\text{UB}/\sigma$, the number of jobs per machine is at most $\sigma/\varepsilon + 1 \leq \sigma\Sigma/\varepsilon + 1$. □

Note that in the proof we showed that the number of jobs per machine in an optimal solution is upper bounded by $\sigma/\varepsilon + 1$. Thus, the set K contains more configurations than are really needed for our instance. Nonetheless, the overestimation of the number of jobs is necessary so that, when a new job arrives and the upperbound is increased (by at most a factor of 2), the set K will still contain all necessary configurations.

We can now describe a schedule of N as a vector $(x_k)_{k \in K}$, such that x_k denotes the number of machines that obey configuration k in the schedule. Then,

it is easy to see that the optimal solution to N found in Lemma 7 corresponds to a vector x that belongs to the solution set of the following set of constrains:

$$\sum_{k \in K} x_k = m, \tag{4}$$

$$\sum_{k \in K} k(i) x_k = n_i \qquad \text{for all } i \in I(\text{UB}), \tag{5}$$

$$x_k \in \mathbb{Z}_{\geq 0} \qquad \text{for all } k \in K.$$

We denote by $A = A(K, I)$ the matrix defining the set of equations (4) and (5); its corresponding right-hand-side is denoted by $b(N, m)$. Then, the non-negative integral solutions to equations (4) and (5) correspond to the set $S = \left\{x \in \mathbb{N}_0^K | Ax = b(N, m)\right\}$. A key point in the following argument is that the set S is embedded in a constant dimensional space, i. e., $S \subseteq \mathbb{Z}^K$, where $|K| \in 2^{O(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})}$.

4.2 Constructing stable solutions

In the following we present the main structural contribution of this paper: We show how to obtain a robust optimal solution to N such that, upon arrival (or departure) of a new job of size at least $\varepsilon \cdot OPT$, we need to migrate jobs of total processing time at most $f(\varepsilon) \cdot OPT$ in order to maintain optimality. This implies that the migration factor needed for this case is upper bounded by $f(\varepsilon)/\varepsilon$. Let us order and relabel the set of configurations $K = \{k_1, \ldots, k_{|K|}\}$ in non-decreasing order of their load, i. e., $\text{load}(k_1) \leq \text{load}(k_2) \leq \ldots \leq \text{load}\left(k_{|K|}\right)$.

Definition 2. *Let $x, x' \in S$. We say that x' is lexicographically smaller than x, denoted $x' \prec_{lex} x$, if $x_k = x'_k$ for all $k \in \{k_1, \ldots, k_q\}$, and $x'_{k_{q+1}} < x_{k_{q+1}}$, for some $q \in \{0, 1, \ldots, |K| - 1\}$.*

It is easy to see that $\prec_{lex}$ defines a total order on the solution set S, and thus there exists a unique lexicographically minimum vector, which we call x^*. We will soon see that x^* has the proper structure that we need for our purposes. In particular, it maximizes the minimum machine load of instance N.

Lemma 8. *Let x^* be the lexicographically minimum vector in S. Then x^* represents an optimal schedule for instance N.*

Proof. Consider an optimum schedule as in Lemma 7. We can describe such a solution by variables x_k^{OPT} denoting the number of machines that obey configuration $k \in K$. Clearly, $x_k^{OPT} = 0$ for all k such that $\text{load}(k) < OPT$. Thus, since x^* is lexicographically minimum, $x_k^* = 0$ for all $k \in K$ with $\text{load}(k) < OPT$ (since otherwise $x^{OPT} \prec_{lex} x^*$). The result follows. □

Moreover, x^* can be computed in polynomial time by solving a sequence of integer linear programs in constant dimension. For this consider the following algorithm.

ALGORITHM MINLEX

1. Solve $\min\{x_{k_1} | x \in S\}$ using Lenstra's algorithm [10] and call the optimum objective value $x^*_{k_1}$.
2. For each $q = 2, \ldots, |K|$, fix $(x_{k_1}, \ldots, x_{k_{q-1}})$ to $(x^*_{k_1}, \ldots, x^*_{k_{q-1}})$ and compute $x^*_{k_q}$ by minimizing x_{k_q} over S, i.e., compute

$$x^*_{k_q} := \min\left\{x_{k_q} \,\middle|\, x \in S \text{ and } x_{k_r} = x^*_{k_r} \text{ for all } r = 1, \ldots, q-1\right\}.$$

3. Return x^*.

By a simple inductive argument, we can show that this greedy algorithm finds the lexicographically minimum element in S. Alternatively, we can find x^* by solving a single ILP in constant dimension. This can be achieved by minimizing a carefully chosen linear function over the set S. Let $\lambda := 1/(m+1)$, and define $c_q := \lambda^q$ for all $q \in \{1, \ldots, |K|\}$. Consider the following problem, which we denote by [LEX].

$$\min\left\{\sum_{q=1}^{|K|} c_q x_{k_q} : A \cdot x = b(N, m) \text{ and } x_k \in \mathbb{Z}_{\geq 0} \text{ for all } k \in K\right\}.$$

Lemma 9. *Let z be an optimal solution to [LEX]. Then, z is the lexicographically minimal solution in S.*

Proof. We use the following claim which is obtained by standard calculus techniques.

Claim. For each $\ell \in \{1, \ldots, |K| - 1\}$, it holds that $m \cdot \sum_{q=\ell+1}^{|K|} c_q < c_\ell$.

Let z be an optimum solution to [LEX] and x^{lex} the lexicographically minimum solution in S. We proceed by contradiction, and call ℓ the smallest index such that $z_{k_\ell} \neq x^{lex}_{k_\ell}$. Since x^{lex} is the lexicographically minimum solution, we know that $x^{lex}_{k_\ell} \leq z_{k_\ell} - 1$. Then,

$$\begin{aligned}\sum_{q=\ell}^{|K|} c_q x^{lex}_{k_q} &\leq c_\ell(z_{k_\ell} - 1) + \sum_{q=\ell+1}^{|K|} c_q x^{lex}_{k_q} \\ &\leq c_\ell(z_{k_\ell} - 1) + \sum_{q=\ell+1}^{|K|} c_q(z_{k_q} + m) < \sum_{q=\ell}^{|K|} c_q z_{k_q},\end{aligned}$$

where the last inequality follows from the claim above. Finally, adding

$$\sum_{q=1}^{\ell-1} c_q x_{k_q}^{lex} \;=\; \sum_{q=1}^{\ell-1} c_q z_{k_q}$$

on both sides yields a contradiction to the optimality of z. □

With this last result we can already describe an offline PTAS for our problem: compute an upper bound UB; define vector N by equations (1), (2), and (3); and solve [LEX] with Lenstra's algorithm. Since we can compute UB in linear time (see Appendix A for details), the running time of this algorithm is in $O(n)$.

Let S be the schedule constructed from z. We next show that S is robust. Indeed, we can see that if we slightly change the right-hand-side of [LEX], the new job j^* can be incorporated into the ILP, allowing us to solve this new instance. Indeed, as discussed before, we can assume that p_{j^*} is a power of $(1+\varepsilon)$ and is larger than $\varepsilon \cdot OPT \geq \varepsilon \text{UB}/\sigma$ (by taking $\sigma \geq \gamma$). Then, we can round the new instance $\mathcal{I}'$ by defining a vector $N' = (n'_i)_{i \in I}$ as follows:

$$n'_i = \begin{cases} n_i & \text{if } p_{j^*} \neq (1+\varepsilon)^i \\ n_i + 1 & \text{if } p_{j^*} = (1+\varepsilon)^i, \end{cases} \qquad \text{for } i = \ell, \ldots, u-1,$$

$$n'_u = \begin{cases} n_u + 1 & \text{if } p_{j^*} \geq (1+\varepsilon)^u \\ n_u & \text{otherwise.} \end{cases}$$

In other words, if $p_{j^*} \geq (1+\varepsilon)^u \geq 4A \geq A' \geq OPT'$, then job j^* is processed on a machine of its own, and thus we can assume that its size is just $(1+\varepsilon)^u$. Otherwise, we increase the number of jobs of size p_{j^*} by one. Also, note that all jobs whose size was rounded down to $(1+\varepsilon)^u$ in the original instance $\mathcal{I}$ are still larger than $\Sigma \cdot \text{UB} \geq \text{UB}'$, and thus get a machine of their own in OPT'. Moreover, jobs that are smaller than $\varepsilon \cdot \text{UB}/\sigma$ are also smaller than $\varepsilon \cdot OPT'/\sigma$. Thus, using the same argument as in Lemma 6, solving instance N' yields a $(1 - O(\varepsilon))$-approximate solution to $\mathcal{I}'$. Also, analogously to Lemmas 7 and 9, we can solve this instance by optimizing the following modification of [LEX], which we call [LEX]':

$$\min \left\{ \sum_{q=1}^{|K|} c_q x_{k_q} : Ax = b(N', m) \text{ and } x_k \in \mathbb{Z}_{\geq 0} \text{ for all } k \in K \right\}.$$

Let z' be an optimum solution to this problem. Note that this ILP is the same as [LEX] except that one of the entries of the right-hand-side is modified by one. With the last observation plus a sensitivity analysis result, we are able to bound the difference between z and z', and thus conclude our claim.

Theorem 1. *There exists a static robust PTAS if the arriving job is larger than* $\varepsilon \cdot OPT$.

Proof. It follows from a sensitivity analysis result[3], and the fact that lexicographically minimal solutions are unique, that

$$\|z - z'\|_\infty \leq |K|\Delta \left(\left\| b(N,m) - b(N',m) \right\|_\infty + 2 \right) \leq 3|K|\Delta \,,$$

where Δ is the maximum over all the subdeterminants of A in absolute values. Using the same argument as in [12], we can easily check that $\Delta \in 2^{O(\frac{1}{\varepsilon}\log^2 \frac{1}{\varepsilon})}$, and therefore

$$\|z - z'\|_1 \leq 3|K|^2\Delta \in 2^{O(\frac{1}{\varepsilon}\log^2 \frac{1}{\varepsilon})} \,.$$

We thus need to touch at most $2^{O(\frac{1}{\varepsilon}\log^2 \frac{1}{\varepsilon})}$ machines to get from the solution given by z to the one given by z'. Also, all jobs that are migrated have (unrounded) processing times at most $(1+\varepsilon)^u$, and must start processing before $(1+\varepsilon)^u$. Thus, the total processing time on each of these machines is at most $2\Sigma \cdot \text{UB}$, and since $p_{j^*} \geq \varepsilon \cdot OPT \geq \varepsilon \cdot \text{UB}/\gamma$, the migration factor is upper bounded by $6\Sigma\gamma|K|^2\Delta/\varepsilon \in 2^{O(\frac{1}{\varepsilon}\log^2 \frac{1}{\varepsilon})}$. □

Running time. By the last theorem, given z we can compute z' by an exhaustive search through all vectors feasible to [LEX]', and whose components differ from z by at most $2^{O(\frac{1}{\varepsilon}\log^2 \frac{1}{\varepsilon})}$. Therefore, the running time needed to compute z', and thus the solution to $\mathcal{I}'$, is $2^{2^{O(\frac{1}{\varepsilon}\log^2 \frac{1}{\varepsilon})}}$.

Job departure. We can adjust the techniques presented in this section for the job departure case. Indeed, if instance $\mathcal{I}'$ contains one less job than $\mathcal{I}$, we can assume that $\mathcal{I}'$ is nontrivial since the trivial case can be dealt separately as in Lemma 5. Therefore $\text{UB}' \geq \text{UB}/2$, and thus taking σ larger than 2, implies that $I(\text{UB})$ contains all jobs sizes between $\varepsilon \cdot OPT'$ and OPT'. This implies that rounding instance $\mathcal{I}'$ within this range decreases the optimum value by at most a factor $(1 - O(\varepsilon))$. Moreover, the right hand side of [LEX] and [LEX]' differ in only one entry, where [LEX]' has one component decreased by one. Then, the same sensitivity analysis result can be used to conclude that the migration factor needed to construct the solution given by [LEX]' is at most $2^{O(\frac{1}{\varepsilon}\log^2 \frac{1}{\varepsilon})}$.

[3] **Lemma**. (*from [13]*) Let A be an integral $m \times n$-matrix, such that each subdeterminant is at most Δ in absolute value, let b and b' be column m-vectors, and let c be a row n-vector. Suppose $\min\{cx | Ax \leq b; x \in \mathbb{Z}^n\}$ and $\min\{cx | Ax \leq b'; x \in \mathbb{Z}^n\}$ are finite. Then, for each optimum solution z for the first problem there exists an optimum solution z' of the second problem such that $\|z - z'\|_\infty \leq n\Delta(\|b - b'\|_\infty + 2)$.

5 Maintaining robust solutions dynamically

In the previous section we showed how to construct a robust $(1-O(\varepsilon))$-approximate solution, so that we can maintain the approximation guarantee at the arrival (departure) of an arbitrary new big job and keep the migration factor bounded. Nonetheless, we cannot further iterate this method when more jobs arrive or depart. Notice that in the last section, we chose the range on which we round our jobs (i. e., set $I(\text{UB})$) large enough so to ensure that the optimum of the rounded instances N and N' are close to the optimum of the original instances. Nonetheless, as more jobs arrive (depart), the optimal value of the new instances may become arbitrarily large (small), and thus the range $I(\text{UB})$ will not be large enough to guarantee the approximation ratios of the rounded instances. On the other hand, we cannot make the index set $I(\text{UB})$ larger so as to simultaneously round all possible instances and maintain the number of job sizes constant.

We deal with this difficulty by dynamically adjusting the set $I(\text{UB})$, defining it appropriately for the current instance. In doing so, we must be extremely careful not to destroy the structure of the constructed solutions and maintain the reassignment cost bounded. In particular, notice that each time that set $I(\text{UB})$ is shifted to the right (left), we must regroup small jobs into larger (smaller) groups. Then, we should avoid changing $I(\text{UB})$ too often: it should be changed only when we can guarantee that there is enough reassignment potential accumulated to regroup all small jobs and simultaneously maintain the structure of optimal solutions. To this end we propose the following algorithm. Let $\mathcal{I}$ be the instance after the t-th job arrival/departure. We run the following algorithm on instance $\mathcal{I}$ for every t:

ROBUST PTAS

1. Run algorithm STABLE-AVERAGE over $\mathcal{I}$ to compute A and w.
2. If variable A_0 has not yet been defined or $A \notin [A_0/2, 2A_0]$, then (re)define $\mathcal{I}_0 \leftarrow \mathcal{I}$ and $A_0 \leftarrow A$.
3. Using as an upperbound $\text{UB} = 2A_0$, define sets $I(\text{UB})$, K, and the ILP [LEX] as in the previous section. Compute the optimal solution z to [LEX], and construct a schedule S using z as a template.

Notice that throughout the algorithm, $OPT \leq A \leq 2A_0 = \text{UB}$, and thus UB is indeed an upperbound on OPT. Moreover, $OPT \geq A/2 \geq A_0/4 = \text{UB}/8$, and thus UB is within a factor 8 of OPT (i.e., $\gamma = 8$ with the notation of the previous section). By the discussion in Section 4.2, an appropriate choice of values σ and Σ guarantees that all constructed solutions are $(1 - O(\varepsilon))$-approximate. We just have to show that the needed reassignment factor is bounded by a constant.

For the analysis we separate the iterations of the algorithm into blocks. Each block B consists of a consecutive sequence of instances, so that the value of A_0 in the algorithm is kept constant within B. Thus, for each instance $\mathcal{I}$ occurring in B, its stable average A belongs to the interval $[A_0/2, 2A_0]$. Consider two consecutive instances $\mathcal{I}$ and $\mathcal{I}'$ that belong to the same block. In what follows we add a symbol *prime* to denote the variables corresponding to $\mathcal{I}'$. So, for example, A' denotes the stable average of $\mathcal{I}'$. Since $\mathcal{I}$ and $\mathcal{I}'$ belong to the same block B, we have that $\text{UB} = 2A_0 = \text{UB}'$, and thus the sets $I(\text{UB})$ and K coincide with $I(\text{UB}')$ and K', respectively. Therefore, the ILPs [LEX] and [LEX]' only differ in their right-hand-side, where one entry is changed by at most a value of one. With this observation we can use the same reasoning as in Theorem 1 to prove the following lemma.

Lemma 10. *If two consecutive instances $\mathcal{I}$ and $\mathcal{I}'$ belong to the same block of iterations, then the migration factor used to obtained S' from S is at most $2^{O(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})}$.*

It remains to consider the limit case where instance $\mathcal{I}$ and the next instance $\mathcal{I}'$ belong to different blocks. For this, we assume that $\mathcal{I}'$ contains one more job than $\mathcal{I}$ and that $A' > 2A_0$. The case where $A' < A_0/2$ can be dealt in an analogous way.

If $\mathcal{I}$ is trivial, it is easy to see that the constructed schedule S is the optimal solution to the instance. Moreover, by the same reasoning as in Lemma 5, schedules S and S' will not process two jobs larger than $\Sigma \cdot \text{UB}$ on the same machine. Then, up to permutation of machines, S and S' only differ by jobs smaller than A. Since the processing time of all such jobs is A and the arriving job is larger than $\varepsilon \cdot OPT \geq \varepsilon \cdot A/2$, we conclude that the migration factor needed for this iteration is at most $2/\varepsilon$.

We now take care of the case where $\mathcal{I}$ is nontrivial. Assume that $\mathcal{I}$ belongs to a block B, and consider the value of A_0 corresponding to this block. It holds that $A_0 \leq A \leq 2A_0$. Also, since $\mathcal{I}$ is nontrivial, Lemma 1 ensures that $A \leq A' \leq 2A$, and therefore $\text{UB} \leq \text{UB}' \leq 4\text{UB}$.

To be able to compare solutions $z \in \mathbb{N}_0^K$ and $z' \in \mathbb{N}_0^{K'}$, we first need to interpret them in a common euclidian space containing them. For this purpose, notice that huge jobs of $\mathcal{I}$ have processing time larger than $\Sigma \cdot \text{UB} \geq \text{UB}'$, assuming that Σ is chosen larger than $4(1+\varepsilon)$. Thus, these jobs take a machine of their own in solutions OPT, OPT', S, and S', and thus we do not need to consider them. We can then assume that all jobs of $\mathcal{I}$ and $\mathcal{I}'$ have processing time upper-bounded by $\Sigma \cdot \text{UB}$. In particular, the entries of vector $N' = (n'_i)_{i \in I(\text{UB}')}$ are zero if $(1+\varepsilon)^i > \Sigma \cdot \text{UB}$. We can then interpret N' as a vector in $\mathbb{N}_0^I$ by setting to zero the entries n'_i with $(1+\varepsilon)^i < \varepsilon \cdot \text{UB}'/\sigma$.

With this simplification, we note that the configurations $k' \in K'$ that will be used by our solution z, must satisfy $k'(i) = 0$, for all $i \in I' \setminus I$. Also, arguing as in Lemma 7, if Σ is large enough, configurations in K contain enough jobs to describe optimal solutions to N'. Thus, configurations $k' \in K'$ with $z'_{k'} \neq 0$ can be interpreted as configurations in K. We conclude that vector z' can be regarded as a vector in $\mathbb{N}_0^K$. Moreover, this vector must be the solution to [LEX] when the right-hand-side is changed from N to N' (when N' is interpreted as a vector in $\mathbb{N}_0^I$). In what follows we bound the difference between N' and N in terms of the number of jobs that have arrived in block B. This will then let us bound the reassignment factor needed by our algorithm.

Lemma 11. *Let q be the number of jobs that have arrived in block B, including the job that made the algorithm change to the next block. Then, $\|N - N'\|_1 \in O(q/\varepsilon)$.*

For simplicity, we brake the proof into several lemmas. Recall that $I(\text{UB}) := \{\ell, \ldots, u\}$, and also $I(\text{UB}') := \{\ell', \ldots, u'\}$. As discussed before, for our purposes we can assume that $u = u'$, and that $\ell < \ell'$. We must then upper-bound $\sum_{i=\ell}^{u} |n_i - n'_i|$. We brake this sum into parts and bound them separately.

Lemma 12. *Vectors N and N' satisfy $\sum_{i=\ell'+1}^{u} |n_i - n'_i| \leq 1$.*

Proof. Note that no jobs of processing time $(1+\varepsilon)^i$, with $i \in \{\ell'+1, \ldots, u\}$, were ever rounded or grouped, and thus the only difference between n_i and n'_i can be due to the newly arriving job. □

It remains to bound $\sum_{i=\ell}^{\ell'} |n_i - n'_i|$. For this, note that $n_{\ell'}$ and $n'_{\ell'}$ can only differ due to the newly arrived job, or the jobs that were smaller than $(1+\varepsilon)^{\ell'}$ and were grouped into jobs of size $(1+\varepsilon)^{\ell'}$. Moreover, the number of the regrouped jobs can be no more than the number of jobs before grouping. Therefore,

$$|n_{\ell'} - n'_{\ell'}| \leq \sum_{i=\ell}^{\ell'-1} n_i + 1,$$

and thus,

$$\sum_{i=\ell}^{\ell'} |n_i - n'_i| \leq 2 \cdot \sum_{i=\ell}^{\ell'-1} n_i + 1.$$

Lemma 13. *The expression $\sum_{i=\ell}^{\ell'-1} n_i$ is in $O(q/\varepsilon)$.*

Proof. Recall that block B is the block containing instance $\mathcal{I}$. Let $\mathcal{I}_0$ be the first instance of block B (as it was also defined in the algorithm), and let N_0 be its

corresponding rounded instance. Also, let A_0 and w_0 be the values returned by Algorithm STABLE-AVERAGE when applied to $\mathcal{I}_0$.

Recall that q is the number of jobs arriving between $\mathcal{I}_0$ and $\mathcal{I}'$. If x is the number of jobs smaller than $(1+\varepsilon)^{\ell'}$ in N_0, then $\sum_{i=\ell}^{\ell'-1} n_i \leq x+q$. We can find an upperbound on x by noting that Lemma 4 implies $w_0/2 < q$. On the other hand, in any optimal solution to instance $\mathcal{I}_0$, each job smaller than A_0 must be processed on one of the w_0 machines not containing huge jobs. Therefore, the volume of such jobs is bounded by $2A_0w_0$. Since all (grouped) jobs are larger than $\varepsilon \cdot \text{UB}/\sigma = 2\varepsilon \cdot A_0/\sigma$, it holds that $x \leq \sigma w_0/\varepsilon \leq 2\sigma q/\varepsilon$. We finally conclude that $\sum_{i=\ell}^{\ell'-1} n_i \leq x + q \leq q(2\sigma/\varepsilon + 1)$. □

Proof (Lemma 11). Collecting the inequalities from the last two lemmas, we obtain that

$$\sum_{i\in\ell}^{u} |n_i - n'_i| \;\leq\; 2 \cdot \sum_{i=\ell}^{\ell'-1} n_i + 2 \;\leq\; 2q(2\sigma/\varepsilon + 1) + 2 \;\in\; O(q/\varepsilon)\,.$$

This concludes the proof. □

Applying Lemma 11 and the same proof technique as in Theorem 1, we obtain the following lemma.

Lemma 14. *The reassignment potential used to construct S' bounded from above by $q \cdot A_0 \cdot 2^{O(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})}$.*

Collecting all our results, we can conclude the main result of this paper.

Theorem 2. *For the machine covering problem with jobs arriving and departing online, there exists a $(1+\varepsilon)$-competitive polynomial algorithm with constant reassignment factor at most $2^{O(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})}$.*

Proof. Consider a fixed block of iterations, and let κ be the number of arriving/departing jobs in this block, including the job that made the algorithm switch to the next block. Lemmas 10 and 11 assure that the total reassignment potential used during this block and to change to the next block is at most $A_0\kappa 2^{O(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})}$, because $q \leq \kappa$. Since all arriving jobs are larger than $\varepsilon \cdot OPT \geq \varepsilon \cdot A_0/4$, we conclude that the reassignment factor needed in this block is at most $2^{O(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})}$. Moreover, we did not use any reassignment potential accumulated from previous blocks. We conclude that the reassignment factor of the algorithm is at most $2^{O(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})}$. □

6 Reducing the accumulated reassignment potential

We will devote this chapter to reduce the reassignment potential needed in the algorithm, so that is always bounded by $O(\varepsilon) \cdot OPT$. This is, as justified by Lemma 1, best possible. Notice that in the Algorithm ROBUST PTAS, an iteration that produces a change of blocks may use the reassignment potential accumulated during the whole previous block. Thus, the migration factor of this iteration is not bounded. We can refine the algorithm so that the migration factor is always bounded by a constant if arriving/departing jobs are larger than $\varepsilon \cdot OPT$. Therefore, we only need to accumulate reassignment potential due to the small jobs as explained in Section 2. It is easy to see that the reassignment potential accumulated due to small arriving jobs is at most $O(\varepsilon) \cdot OPT$. This, however, does not hold for small jobs that leave the system, since we may need to accumulate the reassignment potential of jobs with total processing time up to $O(\varepsilon m) \cdot OPT$. Thus, we deal with departing small jobs similarly as to the job arriving case. We ignore the fact that jobs smaller than $\varepsilon \cdot OPT$ leave until their total size has surpassed $\varepsilon \cdot OPT$. Only then we remove this set of jobs all at once. For presentation issues, we will first assume that all arriving/departing jobs are larger than $\varepsilon \cdot OPT$. We will clarify at the end of this section how to deal with the case of small leaving jobs.

Recall that Lemma 14 shows that the accumulation of the reassignment potential on the interface between block B and the following block is in $O(\kappa) \cdot OPT$, where κ is the number of iterations in B. Moreover, the reassignment potential is only needed for regrouping jobs that were smaller than $\varepsilon \mathrm{UB}'/\sigma$. The refinement of our algorithm works by regrouping small jobs along all the κ iterations corresponding to B.

Consider an instance $\mathcal{I}$ and, as before, let N be its corresponding rounded instance, defined over the set of job size indeces $I(\mathrm{UB}) = \{\ell, \ldots, u\}$. We will construct N in a slightly different way as before: small jobs are not necessarily grouped into jobs of size $(1+\varepsilon)^\ell$, but possibly into larger groups. To this end we will additionally maintain a vector $N^{gr} = (n_i^{gr})_{i \in I(\mathrm{UB})}$, where n_i^{gr} denotes the number of jobs of size $(1+\varepsilon)^i$ in N that correspond to grouped small jobs. We will do so such that $n_i^{gr} = 0$ if $(1+\varepsilon)^i > (1+\varepsilon)^{\ell^*}$, where $(1+\varepsilon)^{\ell^*} \in O(\varepsilon \cdot OPT)$. This fact is enough to show that the rounding make us loose at most a $1 - O(\varepsilon)$ factor in the objective function.

The following algorithm runs over instance $\mathcal{I}$ whenever a big job arrives or departs. During the algorithm we will use a (big enough) parameter $C \in O(1/\varepsilon)$ that we will choose appropriately later.

ROBUST PTAS II

1. Run algorithm STABLE-AVERAGE over $\mathcal{I}$ to compute A and w.
2. If variable A_0 has not yet been defined or $A \notin [A_0/2, 2A_0]$, then (re)define $\mathcal{I}_0 \leftarrow \mathcal{I}$ and $A_0 \leftarrow A$.
3. Using as an upperbound $\text{UB} = 2A_0$, define set $I(\text{UB}) = \{\ell, \ldots, u\}$.
4. If $A \geq A_0$:
   (a) Define $\ell^* := \left\lceil \log_{1+\varepsilon} \frac{4\varepsilon \text{UB}}{\sigma} \right\rceil$.
   (b) Let $\widehat{N}$ be the rounded instance of the previous iteration. Construct N by first taking $\widehat{N}$ and adding (removing) the newly arriving (departing) job.
   (c) Let T be the number of jobs with size less than $(1+\varepsilon)^{\ell^*}$ in N. Remove $\min\{C, T\}$ (where C is a constant that will be chosen later) many of these jobs, and regroup them into jobs of size $(1+\varepsilon)^{\ell^*}$. Adjust n_{ℓ^*} so that
   $$0 \leq \sum_{j: p_j \leq (1+\varepsilon)^{\ell^*}} p_j - \sum_{i=\ell}^{\ell^*} n_i (1+\varepsilon)^i \leq (1+\varepsilon)^{\ell^*}. \tag{6}$$
5. If $A < A_0$:
   (a) Let $\widehat{N}$ be the rounded instance of the previous iteration, and $\widehat{N}^{gr}$ its corresponding subinstance of grouped jobs. Construct N by taking $\widehat{N}$ and adding (removing) the newly arriving (departing) job.
   (b) Let T be the number of jobs in $\widehat{N}^{gr}$ of size strictly larger than $(1+\varepsilon)^{\ell}$. Regroup $\min\{C, T\}$ many of these jobs into jobs of size $(1+\varepsilon)^{\ell}$, and adjust n_ℓ so that Equation (6) holds.
6. Define the configuration set K and the ILP [LEX] with right-hand-side $b(N, m)$. Compute the optimal solution to [LEX], z, and construct it associated vector N.

As before, note that within a block (i. e., a sequence of iterations where A_0 remains constant) instance N is changed by adding or removing a constant number of jobs. Thus, we can observe that iteratively applying C times the argument of Theorem 1 we obtain that we need at most a migration factor of $C \cdot 2^{O(\frac{1}{\varepsilon}\log^2 \frac{1}{\varepsilon})}$. Let us consider now the case where $\mathcal{I}$ is the last instance of some block B and that $A > A_0$. As before, we can assume without loss of generality that $\mathcal{I}$ is nontrivial.

Lemma 15. *Let $\mathcal{I}$ be the last instance of a block. Assume that $\mathcal{I}$ is nontrivial and that $A > A_0$. Then, we can choose parameter $C \in O(1/\varepsilon)$ large enough so that instance N only contains jobs larger or equal than $(1+\varepsilon)^{\ell^*}$.*

Proof. Let $\bar{\mathcal{I}}$ be the last instance in B so that its stable average $\bar{A}$ satisfies $\bar{A} \leq A_0$. Also, let q be the number of jobs arriving between instances $\bar{\mathcal{I}}$ and $\mathcal{I}$. By Lemma 4 we know that $q + 1 \geq 2\bar{w}$. Let $\bar{N}$ be the rounded instance of $\bar{\mathcal{I}}$ given by the algorithm, and consider all jobs smaller than $(1+\varepsilon)^{\ell^*}$ in $\bar{N}$. It is clear that the total volume of these jobs can be at most $2\bar{A}\bar{w} \leq 2A_0\bar{w}$, and thus there are at most $2A_0\bar{w}/(1+\varepsilon)^{\ell^*} \in O(q/\varepsilon)$. Thus, if we choose $C \in O(1/\varepsilon)$ large enough, after q iterations of the algorithm we can remove all jobs smaller than $(1+\varepsilon)^{\ell^*}$ and transform them into jobs of size $(1+\varepsilon)^{\ell^*}$. □

Then, if $\mathcal{I}'$ is the instance after $\mathcal{I}$, we can interpret N and N' in the same euclidian space as in last section. The previous lemma ensures that instance N only contains jobs of size $(1+\varepsilon)^i$ where $i \in I(\text{UB})$, and that N and N' differ in at most a constant number of jobs. Therefore, since all jobs in N are larger than $(1+\varepsilon)^{\ell^*} \geq \varepsilon \text{UB}'/\sigma$, the migration factor necessary to transform their corresponding optimal solutions must be bounded by $2^{O(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})}$. We can similarly deal with the case $A < A_0$. Note that in this case the next instance after $\mathcal{I}$, instance $\mathcal{I}'$, must contain one job less than $\mathcal{I}$.

Lemma 16. *Assume that $\mathcal{I}$ is nontrivial, is the last instance of its block and that $A < A_0$. Then, we can choose parameter $C \in O(1/\varepsilon)$ large enough so that all grouped jobs in N have size $(1+\varepsilon)^\ell$, i.e., $n_i^{gr} = 0$ for all $i \in \{\ell+1, \ldots, u\}$.*

Proof. Let $\bar{\mathcal{I}}$ be the last instance in B so that its stable average $\bar{A}$ satisfies $\bar{A} \geq A_0$. Also, let q be the number of jobs leaving between instances $\bar{\mathcal{I}}$ and $\mathcal{I}$. By Lemma 4 we know that $q + 1 \geq 2\bar{w}$. Moreover, consider all jobs smaller than $(1+\varepsilon)^{\ell^*}$ in $\bar{N}$. Recall that all jobs that are grouped in $\bar{N}$ are smaller than $(1+\varepsilon)^{\ell^*}$. It is clear that the total volume of these jobs can be at most $2A_0\bar{w}$, and thus there are at most $2A_0\bar{w}/(1+\varepsilon)^{\ell^*} \in O(q/\varepsilon)$ such jobs. If we choose $C \in O(1/\varepsilon)$ large enough, then q iterations are enough to remove all jobs larger than $(1+\varepsilon)^\ell$ in $\bar{N}^{gr}$ and transformed them into jobs of size $(1+\varepsilon)^\ell$. □

With these two lemmas we directly conclude the following theorem.

Theorem 3. *If all arriving and departing jobs are larger than $\varepsilon \cdot OPT$, then there exists a robust PTAS with constant migration factor. In particular, no reassignment factor is ever accumulated.*

It remains to discuss how to adapt our algorithm for the case of small departing jobs. As said before, we only remove jobs smaller than $\varepsilon \cdot OPT$ when their total processing time surpasses $\varepsilon \cdot OPT$. We can easily modify our algorithm to work for the case when a set of jobs smaller than $\varepsilon \cdot OPT$ leaves the system together. Indeed, we just need to make sure that in steps 4.b and 5.a we remove

all large jobs from instance $\bar{N}$ (which can only be constantly many), and then adjust N so that Equation (6) still holds for the new instance. It is easy to see that with these modifications instance N is within a $(1 - O(\varepsilon))$ factor of OPT, and that the reassignment factor is also bounded as before.

Theorem 4. *There exists a robust PTAS for the machine covering problem that accumulates at most $O(\varepsilon) \cdot OPT$ reassignment potential.*

7 The minimum makespan problem and beyond

We now briefly discuss how to use our ideas to derive a robust PTAS for the minimum makespan problem in the constant reassignment factor scenario. Most of the techniques derived for the machine covering problem directly carry over, and therefore we only point out the main differences. Notice that we can also use the stable average A of the instance as a lower bound on the minimum makespan. Although A will not be within a constant factor of OPT, it still holds that removing jobs that are larger than A preserves optimal solutions. Then, we can define $\text{UB} := A$ (which in this case is a lower bound), the set of indices $I(\text{UB})$ as in Section 4.2, and round jobs so that the processing times equal $(1+\varepsilon)^i$ for some $i \in I(\text{UB})$. Computing a $(1+\varepsilon)$-approximate solution to the rounded instance yields a nearly optimal solution for the original problem. Analogously, the set of configurations K can be considered in the same way. To construct robust solutions, we can also consider the lexicographically minimal vector $z = (z_k)_{K \in K}$, where z_k denotes the number of machines following configuration k. However, we must revert the order given to the configuration set: we relabel the elements of $K := \{k_1, \ldots, k_{|K|}\}$ so that $\text{load}(k_1) \geq \text{load}(k_2) \geq \ldots \geq \text{load}(k_{|K|})$ and consider the lexicographically minimum solution with respect to this order. It is not hard to see that this solution minimizes as much as possible the number of used configurations with large loads, and therefore it also minimizes the makespan. The same analysis as in Section 4.2 holds for this case, and thus the constructed solution is indeed robust. All techniques of Section 5 also carry over without modifications, since they are mostly based on properties of the stable average A.

Similarly, all our techniques work for a very broad class of problems, where the objective functions solely depend on the load of each machine. This set of problems was first considered by Alon, Azar, Woeginger and Yadid [2]. For some function $f : \mathbb{R}_0^+ \to \mathbb{R}_0^+$, we consider the problem of minimizing $\sum_{i \in M} f(\ell_i)$, where ℓ_i denotes the load of machine i in the corresponding schedule. In the same way we can define a corresponding maximization problems.

For minimization (resp. maximization) problems, we require that f (resp. $-f$) satisfies the following properties:

(i) Function f must be *convex*: for all $0 \leq x \leq y$ and $0 \leq \Delta \leq y - x$, it must hold that $f(x + \Delta) + f(y + \Delta) \leq f(x) + f(y)$.
(ii) For all $\varepsilon > 0$ there exists $\delta = \delta(\varepsilon) > 0$ so that

$$\forall x, y \geq 0, \quad |x - y| \leq x \cdot \delta \quad \text{implies} \quad |f(x) - f(y)| \leq \varepsilon \cdot |f(x)|.$$

Let us remark that, for any $p > 1$, the problem of minimizing the L_p norm of the machine loads falls into this framework.

Notice that the first property assures that in an optimal solution a job cannot have starting time S_j if there is a machine with load less than S_j. This ensures that jobs that are larger than the stable average A will have a machine of their own in optimal solutions. Then we can remove them with their corresponding machines and obtain an equivalent instance. Moreover, this property also implies that the load of all the machines in an optimal solution are within a factor 2 of A (see proof of Lemma 2).

On the other hand, for any given $\varepsilon > 0$, the second property ensures that if we round our instances so that the loads of the machines do not change in more than a $1 + \delta(\varepsilon)$ factor, then the objective function will be within a $1 + \varepsilon$ factor of the original optimal solution. Notice that all our rounding techniques satisfy this property.

With this observations it is easy to apply the techniques derive before to obtain a robust PTAS. The only difference is how to model the objective function of [LEX]. However, this is trivial since the form of the objective functions that we consider depends uniquely on the loads of the configurations we use.

Theorem 5. *For a given function $f : \mathbb{R}_0^+ \to \mathbb{R}_0^+$, there exists a robust PTAS with constant reassignment factor for the problem of minimizing (resp. maximizing) $\sum_{i \in M} f(\ell_i)$ on parallel machines if f (resp. $-f$) satisfies properties (i) and (ii).*

References

1. S. Albers. Online algorithms: a survey. *Mathematical Programming*, 97:3–26, July 2003.
2. N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1:55–66, 1998.
3. M. Andrews, M. Goemans, and L. Zhang. Improved bounds for on-line load balancing. *Algorithmica*, 23:278–301, 1999.
4. Y. Azar, B. Kalyanasundaram, S. Plotkin, K. R. Pruhs, and O. Waarts. On-Line load balancing of temporary tasks. *Journal of Algorithms*, 22:93–110, 1997.
5. M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, Aug. 1973.
6. B. Chen, A. van Vliet, and G. J. Woeginger. Lower bounds for randomized online scheduling. *Information Processing Letters*, 51:219–222, 1994.

7. L. Epstein and A. Levin. A robust APTAS for the classical bin packing problem. In *Automata, Languages and Programming*, pages 214–225. 2006.
8. R. Fleischer and M. Wahl. Online scheduling revisited. *Journal of Scheduling*, 3:343–353, 2000.
9. D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.
10. H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, Nov. 1983.
11. J. F. Rudin III and R. Chandrasekaran. Improved bounds for the online scheduling problem. *SIAM Journal on Computing*, 32:717–735, 2003.
12. P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34:481–498, May 2009.
13. A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester, June 1998.
14. J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Information Processing Letters*, 63:51–55, 1997.
15. J. Sgall. On-line scheduling — a survey. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*, pages 196–231. Springer, Berlin, 1998.
16. J. Westbrook. Load balancing for response time. *Journal of Algorithms*, 35:1–16, Apr. 2000.
17. G. J. Woeginger. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Operations Research Letters*, 20:149–154, May 1997.

A Computing lower bounds in linear time

In this section we refine the algorithm STABLE-AVERAGE to run in linear time. The general idea is to use a binary search approach. For the description of the algorithm we use the following notation. Let $Q := \{q_1, \ldots, q_r\}$ the set of all different processing times, and for each $q \in Q$, we denote $J_{\leq q} := \{j \in J : p_j \leq q\}$. Also, we define,

$$f(q) := \frac{p(J_{\leq q})}{m - |J \setminus J_{\leq q}|}.$$

Notice that $f(q)$ can be equal to ∞ if $m = |J \setminus J_{\leq q}|$. Nonetheless we are only interested in values of $f(q)$ when $|J \setminus J_{\leq q}| < m$. With the notation previously introduced, algorithm Stable-Average can be implemented as follows.

1. Order the processing times so that $q_{i_1} \leq q_{i_2} \leq \ldots \leq q_{i_r}$.
2. For each $k = r, \ldots, 1$, check whether $q_{i_k} \leq f(q_{i_k})$. If it does, define $p^* := q_{i_k}$ and return $f(p^*)$. Otherwise, keep iterating.

It is not hard to see that this algorithm return the same value A as algorithm Stable-Average, i.e., $A = f(p^*)$. To avoid ordering the processing times, and therefore reduce the runnining time of the algorithm, we instead use a binary search approach. Notice that alternatively to this algorithm, we can define p^* as the largest processing time of a job so that $p^* \leq f(p^*)$ and $q > f(q)$ for all $q \in Q$ with $q \geq p^*$. This already suggests the following algorithm.

FAST-STABLE-AVERAGE

1. Initilize $\ell \in Q$ and $u \in Q$ as the smallest and largest values in Q respectively.
2. While $\ell \neq u$:
   (a) Compute the median $\bar{q}$ of the set $\{q : \ell \leq q \leq u\}$. *Note: For a given set of n numbers, we say that its* median *is the $\lfloor (n+1)/2 \rfloor$-th largest number in the set.*
   (b) If $\bar{q} = \ell$ then go to Step (3).
   (c) Compute $f(\bar{q})$.
   (d) If $\bar{q} > f(\bar{q}) > 0$, then redefine $u \leftarrow \bar{q}$. Else, $\ell \leftarrow \bar{q}$.
3. Define $p' \leftarrow \ell$, and return $f(p')$.

We first discuss the correctness of the algorithm, i.e., that $p' = p^*$, and afterward we prove that it can be implemented to run in linear time. For the analisys we assume that $q_1 < q_2 < \ldots < q_r$. Notice that we have not used this fact during the algorithm itself. The crucial observation is to note that if $q_i = p^*$ or $q_i = p'$, the following three properties are satisfied:

(i) $q_i \leq f(q_i)$,

(ii) $q_{i+1} > f(q_{i+1})$,

(iii) $f(q_i) \geq 0$.

Therefore, the correctness of the algorithm follows from the next lemma.

Lemma 17. *There exists a unique value $q \in Q$ satisfying Properties (i), (ii) and (iii).*

To show this lemma we first need the following technical result. Figure 2 shows the claims of this next lemma.

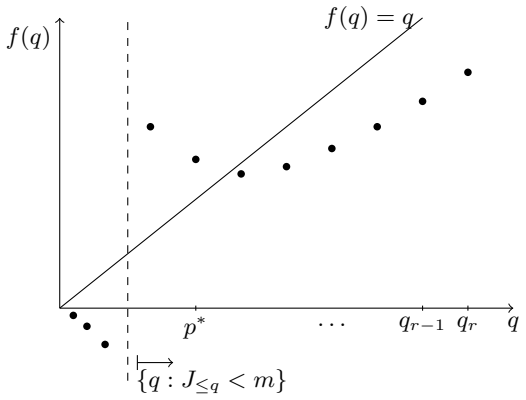


Fig. 2: Behaviour of function f.

Lemma 18. *For each $q_i \in Q$ so that $|J \setminus J_{\leq q_{i-1}}| < m$, we have that $q_i \leq f(q_i)$ if and only if $f(q_i) \leq f(q_{i-1})$.*

Proof. To simplify the computations we introduce the following notation. Let $J_i := \{j \in J : p_j = q_i\}$, $s_i := |J_i|$, and $m_i := m - |J \setminus J_{\leq q_i}|$. Notice that $m_i - s_i > 0$, since $m_i - s_i = m - |J \setminus J_{\leq q_{i-1}}|$. We then have the following

sequence of equivalences

$$\begin{aligned}
f(q_{i-1}) &\geq f(q_i) \\
\frac{p(J_{\leq q_{i-1}})}{m_i - s_i} &\geq \frac{p(J_{\leq q_i})}{m_i} \\
\frac{p(J_{\leq q_i})}{m_i - s_i} &\geq \frac{p(J_{\leq q_i})}{m_i} + \frac{p(J_i)}{m_i - s_i} \\
p(J_{\leq q_i}) \left(\frac{1}{m_i - s_i} - \frac{1}{m_i} \right) &\geq \frac{p(J_i)}{m_i - s_i} \\
p(J_{\leq q_i}) \frac{s_i}{m_i} &\geq p(J_i) = s_i \cdot q_i \\
f(q_i) &\geq q_i.
\end{aligned}$$

□

Proof (Proof of Lemma 17). Consider by contradiction that there exists $q_s, q_t \in Q$ satisfying Properties (i), (ii) and (iii) with $q_s < q_t$. Notice that by Property (i), we have that $q_t \leq f(q_t)$, and thus by the previous lemma, $f(q_{t-1}) \geq f(q_t) \geq q_t > q_{t-1}$. Applying Lemma 18 once again for q_{t-1}, we obtain that $f(q_{t-2}) \geq f(q_{t-1}) \geq q_{t-1} > q_{t-2}$. Iterating this argument, we obtain that $f(q_{s+1}) > q_{s+1}$, which contradicts Property (ii) for q_s. □

We have proved the following theorem.

Theorem 6. *Algorithms* STABLE-AVERAGE *and* FAST-STABLE-AVERAGE *returns the same output.*

Finally, we must show that LOWERBOUND finishes in linear time

Theorem 7. *The running time of Algorithm* LOWERBOUND *is* $O(n)$.

Proof. We show that the k-th iteration of Step (2) of the algorithm can be implemented to run in $O(n/2^k)$. By summing over all iterations this proves the claim. Indeed, since in every iteration we reduce the set $\{q : \ell \leq q \leq u\}$ at least by half, then this set has at most $n/2^k$ elements in the k-th iteration. We conclude that Step (2.a) takes time $O(n/2^k)$ since it is possible to compute the median in linear time [5]. For computing Step (2.b), notice that we have already computed $f(\ell)$ in previous iterations, and thus we assume that we store its values. Moreover, it is clear than computing $P := p(\{q : \ell < q \leq \bar{p}\})$ takes time $O(n/2^k)$. We thus conclude by noting that

$$f(\bar{p}) := \frac{f(\ell)(m - |J \setminus J_{\leq \ell}|) + P}{m - |J \setminus J_{\leq \bar{p}}|}.$$

□