Additive Approximation Schemes for Load Balancing Problems

Moritz Buchem
Maastricht University, Maastricht, The Netherlands

Lars Rohwedder
EPFL, Lausanne, Switzerland

Tjark Vredeveld
Maastricht University, Maastricht, The Netherlands

Andreas Wiese
University of Chile, Santiago, Chile

Abstract

We formalize the concept of *additive approximation schemes* and apply it to load balancing problems on identical machines. Additive approximation schemes compute a solution with an absolute error in the objective of at most ϵh for some suitable parameter h and any given $\epsilon > 0$. We consider the problem of assigning jobs to identical machines with respect to common load balancing objectives like *makespan minimization*, the *Santa Claus* problem (on identical machines), and the *envy-minimizing Santa Claus* problem. For these settings we present additive approximation schemes for $h = p_{\max}$, the maximum processing time of the jobs.

Our technical contribution is two-fold. First, we introduce a new relaxation based on integrally assigning slots to machines and *fractionally* assigning jobs to the slots. We refer to this relaxation as the *slot-MILP*. While it has a linear number of integral variables, we identify structural properties of (near-)optimal solutions, which allow us to compute those in polynomial time. The second technical contribution is a local-search algorithm which rounds any given solution to the slot-MILP, introducing an additive error on the machine loads of at most $\epsilon \cdot p_{\max}$.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Load balancing, Approximation schemes, Parallel machine scheduling

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.42

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2007.09333` [6]

Funding *Lars Rohwedder*: Swiss National Science Foundation project 200021-184656.
Andreas Wiese: Partially supported by FONDECYT Regular grant 1200173.

Acknowledgements We wish to thank José Verschae, Alexandra Lassota and Klaus Jansen for helpful discussions.

1 Introduction

Typically, when constructing an approximation algorithm, one provides a multiplicative approximation ratio ρ such that the respective algorithm finds a solution of value at most (or at least, in case of maximization problems) $\rho \cdot \mathrm{OPT}$, where OPT is the optimal solution value. Then, an approximation scheme is a family with an algorithm for each ratio $\rho = 1 + \epsilon$ with $\epsilon > 0$ (or $\rho = 1 - \epsilon$ for maximization problems).

In this paper, we study approximation algorithms and schemes for which we measure their performance as the absolute difference between the value of their computed solution and OPT. We measure this difference with respect to some problem depending quantity h (which might be much smaller than OPT).

▶ **Definition 1.** *For a given optimization problem, consider a quantity defined by $h(I)$ for each instance I. An algorithm is an* additive approximation algorithm w.r.t. *h if for any instance I it computes in polynomial time a solution with value $A(I)$ satisfying $|A(I) - \mathrm{OPT}(I)| \leq h(I)$. An* additive approximation scheme w.r.t. *h is a family of algorithms containing an additive approximation algorithm w.r.t $\epsilon \cdot h$ for each $\epsilon > 0$.*

Studying additive approximation algorithms is particularly interesting in the following two scenarios.

1. If $h(I) \ll (\rho - 1)\mathrm{OPT}(I)$ for some $\rho > 1$ then an additive approximation algorithm w.r.t. h gives a much stronger guarantee than a (multiplicative) ρ-approximation algorithm. In particular, if $h(I) \ll \mathrm{OPT}(I)$ an additive approximation schemes then gives a much stronger guarantee than a PTAS.
2. When there cannot exist a PTAS, or even any multiplicative guarantee for a given problem, additive approximation algorithms give an alternative notion for approximating it. A notable example is the case when it is NP-hard to decide whether $\mathrm{OPT} = 0$, as then no multiplicative approximation guarantee can be obtained.

While a lot of research has focused on traditional multiplicative approximation guarantees, additive approximation guarantees are relatively unexplored in the literature. Notable exceptions include Vizing's algorithm that finds an edge coloring with at most $\Delta + 1$ colors, where Δ is the maximum degree of a graph [36], which is hence an additive approximation for $h \equiv 1$. Also, for bin packing there is an algorithm known that uses at most $\mathrm{OPT} + O(\log \mathrm{OPT})$ bins [17] (improving earlier results in [25] and [33]) which is thus an additive approximation for $h \equiv O(\log \mathrm{OPT})$.

In this paper, we present additive approximation schemes for scheduling and load balancing problems which are among the classical problems in the literature on approximation algorithms, starting with the seminal work of Graham [14]. In these problems, n jobs need to be processed by m machines such that each job is completely processed by one single machine. Each job j has a given processing time p_j and the load of a machine i is the sum of the processing times of the jobs assigned to i. The goal is to find a schedule (represented by an assignment of jobs to the machines) that optimizes some objective function over the machine loads. Since it is strongly NP-hard to decide whether there is a schedule that assigns the same load to each machine (see [13]), most non-trivial load balancing problems of this form are also strongly NP-hard. This observation has led to extensive research on approximation algorithms. In the following, we consider three variations of load balancing problems.

Objective functions. Our first objective function is to minimize the maximum machine load, i.e., to minimize the *makespan*. This is the one of the most classical scheduling problems on parallel machines and has led to the first approximation algorithms [14,15]. Sahni [34] showed that the problem admits an FPTAS for constant number of machines and Hochbaum and Shmoys [19] found a PTAS if the number of machines is part of the input. Since then, there has been lively research in improving the running time, e.g., to an EPTAS [1, 8, 18, 20, 21].

The second objective function that we consider yields the *max-min allocation* problem [7]. Here, the goal is to maximize the load of the least loaded machine. Bansal and Sviridenko [5] called it the Santa Claus problem when they studied it in the restricted assignment setting. This objective is considered to measure the fairness of the allocation. The case of identical machines was also considered by Woeginger [37] who presents a PTAS.

As a third and final objective function, we consider to minimize the *maximum envy*, which is defined as the maximum load minus the minimum load. This objective has been considered by Lipton et. al [30]. While in the Santa Claus problem fairness is measured

by the minimum load of a machine, in this setting fairness is considered by the difference between the maximum and minimum load. Note that it is strongly NP-hard to decide whether or not the envy is 0. Therefore, unless P = NP, there cannot exist any polynomial time approximation algorithm with any (multiplicative) performance guarantee.

For all three variants there is a simple greedy algorithm, which assigns the jobs iteratively to the respective least loaded machine, and which gives an additive error of $p_{\max} := \max_{j \in J} p_j$. Such an additive approximation w.r.t. $p_{\max}$ exists even for unrelated machines, i.e., when the processing time of a job depends on the machine [29]. Note that this guarantee is incomparable to the error of ϵOPT of a PTAS. In particular, in the regime where $p_{\max} = o(\mathrm{OPT})$ the greedy algorithm is still the best algorithm we know.

Our contribution. In this paper, we present additive approximation schemes w.r.t. $p_{\max}$ for the three load balancing problems defined above on identical machines. For the makespan and Santa Claus objective this gives a significant improvement over the greedy algorithm mentioned above while also dominating the guarantees of the known PTASes[1]; for minimizing the maximum envy this demonstrates how additive approximation schemes can lead to non-trivial guarantees when no multiplicative guarantees are possible.

Since $p_{\max}$ can be much smaller than OPT, the main approach of the known (multiplicative) PTASes does not work when aiming for an additive approximation guarantee of $\epsilon p_{\max}$. In those results, a job j is typically considered as small if $p_j \leq \epsilon\mathrm{OPT}$ and otherwise as large. Then there is only a constant number of large jobs on each machine in the optimal solution which allows methods based on enumeration or integer programming in constant dimension (after rounding or grouping the job sizes according to, e.g., powers of $(1+\epsilon)$). The small jobs are finally assigned greedily. However, if $p_{\max} < \epsilon\mathrm{OPT}$ then these algorithms would simply assign all jobs greedily which yields an additive error of up to $p_{\max} > \epsilon \cdot p_{\max}$. Given this, it seems natural to define a job j to be large if $p_j > \epsilon \cdot p_{\max}$; however, then it is not guaranteed that there are only a constant number of large jobs on each machine and thus the aforementioned techniques are not applicable anymore. Moreover, we cannot even afford to round all job sizes to powers of $1+\epsilon$ since this might yield a total error of $\epsilon \sum_j p_j$, which can be much larger than $\epsilon \cdot p_{\max}$. Therefore, there is need for new (non-trivial) machinery.

To this end, we present a new fractional relaxation for this general class of load balancing problems, which we call the *slot-MILP*. This slot-MILP can be interpreted as a strengthened variant of the assignment-LP. The assignment-LP is a relaxation in which each job is assigned separately (fractionally) to a machine. In the slot-MILP we first group jobs of similar sizes, but we do not round the job sizes (unlike most previous PTASes). In particular, different jobs belonging to the same group may have different job sizes. In addition to the constraints of the assignment-LP we require an *integral* number of jobs of each group to be assigned to each machine (which can be implemented using a linear number of integer variables). Our relaxation can be thought of as assigning slots (for the groups of jobs) *integrally* to machines and then assigning the jobs *fractionally* to the slots.

Due to the many integer variables, it is not obvious how to solve the slot-MILP efficiently. This contrasts our approach to many other approximation algorithms which are based on purely fractional relaxations or on MILP-relaxations with only few integral variables which can be solved with Lenstra's algorithm (e.g., as done in some EPTASes for minimizing the

[1] While for makespan minimization $p_{\max} \leq \mathrm{OPT}$ always holds, for Santa Claus this can be assumed essentially w.l.o.g., since the optimum does not change if job sizes are capped at OPT and the latter can be guessed by binary search.

makespan [20, 21]). Instead, we manage to solve it using non-trivial structural properties combined with dynamic programming. While the additive integrality gap of the assignment-LP can be as large as $p_{\max}$, for the slot-MILP this gap is only $\epsilon \cdot p_{\max}$. We show this using a rounding procedure inspired by a local search method for the restricted assignment problem [23, 24, 35]. The local search algorithm repeatedly moves jobs between machines, eventually converging to a good solution. Although in the restricted assignment problem no polynomial running time bound is known for the local search procedure, in our case we obtain such a bound for our local search.

We remark that our slot-MILP is stronger than the known configuration-LP, e.g., since for minimizing the makespan on identical machines, the latter has a multiplicative integrality gap of at least $1+\frac{1}{1023}$ [28] and hence an additive integrality gap of at least $\mathrm{OPT}\cdot\frac{1}{1023} \geq p_{\max}\cdot\frac{1}{1023}$.

Other related work. The case of small values of $p_{\max}$ has also been considered from a parameterized point of view: If all processing times are integers, then it is possible to obtain a running time that is fixed-parameter tractable (FPT) for the parameter $k = p_{\max}$ [26, 27, 31]. In other words, there is an algorithm that finds an exact solution in time $f(k) \cdot |I|^{O(1)}$ for some computable function f.

Other variants of load balancing problems on identical machines have been considered by Alon et al. [1]. They identify some conditions on the objective function, e.g. makespan minimization and the Santa Claus problem, so that the load balancing or machine scheduling problem admits an (E)PTAS. The Santa Claus problem has been considered in the restricted assignment setting in which each job is only allowed to be processed on a subset of the machines, starting with [5]. In a series of papers [3, 4, 9, 12, 16], the approximation ratio was further and further improved, and the currently best known result is a polynomial time $(4+\epsilon)$-approximation algorithm [9, 10] and an upper bound of $3+\frac{21}{26} \approx 3.808$ for the integrality gap of the configuration-LP [9] (which can be solved in polynomial time to any desired accuracy).

For the bin packing problem, Jansen et al. [22] present an additive approximation algorithm w.r.t $h \equiv 1$ in time exponential in the optimal number of bins plus a polynomial in the number of items to be packed. Ophelders et al. [32] showed that a simple local search algorithm for the so-called Equitable Hamiltonian Cycle finds a solution that is at most 1 away from the optimal solution value. Alon et al. [2] present an additive approximation algorithm w.r.t. $h = \epsilon n^2$ for every $\epsilon > 0$ for the edge deletion problem to obtain a graph with a monotone property. This is hence an additive approximation scheme for $h = n^2$ (which is in fact an upper bound on the minimum number of edges to be deleted).

2 Slot-MILP

We introduce an alternative relaxation for a general class of load balancing problems on identical machines. We first formally define this class of load balancing problems as the *target load balancing problem.*

▶ **Definition 2.** *In the* target load balancing problem *we are given a set of jobs $\mathcal{J}$ with a processing time p_j for each $j \in \mathcal{J}$ and a set of machines $\mathcal{M}$ with values ℓ, u. The goal is to assign each job $j \in \mathcal{J}$ to a machine $i \in \mathcal{M}$ such that every machine load satisfies the target load interval $[\ell, u]$.*

This generalizes the load balancing settings mentioned earlier, e.g., for makespan minimization we choose $\ell = 0$ and $u = T$, where T is a guess on the optimal makespan.

Let $\epsilon > 0$ and assume w.l.o.g. that $1/\epsilon \in \mathbb{N}$. Our task is to either assert that there is no solution for the given instance or to find a solution in which the load of each machine i is in the interval $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$ with $p_{\max} := \max_{j \in J} p_j$. First we partition the jobs into sets $\mathcal{J}_1, \ldots, \mathcal{J}_{1/\epsilon}$, where for $k = 1, ..., 1/\epsilon$ the set $\mathcal{J}_k$ contains all jobs $j \in \mathcal{J}$ with $p_j \in ((k-1)\epsilon \cdot p_{\max}, k\epsilon \cdot p_{\max}]$. We define a new relaxation for this problem in which for each machine i and each $k = 1, ..., 1/\epsilon$ we specify integrally how many jobs from $\mathcal{J}_k$ are assigned to i (one may imagine that this defines slots for jobs from $\mathcal{J}_k$ on i). Then the jobs from $\mathcal{J}_k$ are assigned fractionally to these slots.

$$\begin{aligned} \sum_{i \in \mathcal{M}} x_{i,j} &= 1 && \forall j \in \mathcal{J} \\ \ell \leq \sum_{j \in \mathcal{J}} p_j x_{i,j} &\leq u && \forall i \in \mathcal{M} \\ \sum_{j \in \mathcal{J}_k} x_{i,j} &= y_{i,k} && \forall i \in \mathcal{M}, \forall k \in \{1, \ldots, 1/\epsilon\} \\ x_{i,j} &\geq 0 && \forall j \in \mathcal{J}, i \in \mathcal{M} \\ y_{i,k} &\in \mathbb{N}_0 && \forall i \in \mathcal{M}, k \in \{1, \ldots, 1/\epsilon\} \end{aligned} \tag{1}$$

We refer to this relaxation as the slot-MILP. The integer variables define exactly how many jobs of a type are assigned to a machine but do not imply a specific load based on rounded processing times. The load of a machine is based on an assignment that satisfies the distribution of slots among the machines.

Since the slot-MILP contains $1/\epsilon \cdot |\mathcal{M}|$ integral variables, it is not clear how to solve it in polynomial time. Nevertheless, we present two methods of efficiently solving the slot-MILP. The first method gives an exact solution while the second method gives a solution that slightly violates the target load intervals. Afterwards, we show how to round a fractional solution of the slot-MILP to an integral solution, while violating the load interval $[\ell, u]$ for each machine $i \in \mathcal{M}$ by at most $\epsilon \cdot p_{\max}$.

2.1 Exact solution method for the relaxation

We make use of a structural property to find an exact solution to the slot-MILP. Note that in this case an exact solution is one that satisfies (1). This structure allows us to guess the values of the integral variables in polynomial time and then the remaining problem is only a linear program.

Given a solution (x, y), we write y_i for the $(1/\epsilon)$-tuple $(y_{i,1}, \ldots, y_{i,1/\epsilon})$. Using similar arguments to [11] we show that there exist solutions in which there are not too many different vectors y_i.

▶ **Lemma 3.** *There is a solution (x, y) to the slot-MILP such that for all $i, i' \in \mathcal{M}$ with $y_i \equiv y_{i'} \bmod 2$ it follows that $y_i = y_{i'}$.*

Proof. Let (x, y) be a solution to the slot-MILP and assume that x is the solution which minimizes

$$\sum_{i \in \mathcal{M}} \sum_{k=1}^{1/\epsilon} \|y_{i,k}\|_2. \tag{2}$$

Now suppose toward contradiction that there are i_1, i_2 with $y_{i_1} \equiv y_{i_2} \bmod 2$, but $y_{i_1} \neq y_{i_2}$. We construct a new solution x', which has a lower value of (2). We set $x'_{i,j} = x_{i,j}$ for all $i \notin \{i_1, i_2\}$ and $x'_{i_1,j} = x'_{i_2,j} = (x_{i_1,j} + x_{i_2,j})/2$. In other words, we evenly distribute all jobs between i_1 and i_2. Let us first check that the solution remains feasible. Let $j \in \mathcal{J}$. Then

$$\begin{aligned}\sum_{i\in\mathcal{M}} x'_{i,j} &= x'_{i_1,j} + x'_{i_2,j} + \sum_{i\notin\{i_1,i_2\}} x'_{i,j} \\ &= \frac{x_{i_1,j} + x_{i_2,j}}{2} + \frac{x_{i_1,j} + x_{i_2,j}}{2} + \sum_{i\notin\{i_1,i_2\}} x_{i,j} \\ &= \sum_{i\in\mathcal{M}} x_{i,j} = 1.\end{aligned}$$

For all machines $i \notin \{i_1, i_2\}$ the load does not change and, hence, the load of machine i remains within $[\ell, u]$. For i_1 and i_2, we argue

$$\begin{aligned}\sum_{j\in\mathcal{J}} p_j x'_{i_1,j} = \sum_{j\in\mathcal{J}} p_j x'_{i_2,j} &= \sum_{j\in\mathcal{J}} p_j \frac{x_{i_1,j} + x_{i_2,j}}{2} \\ &= \frac{1}{2}\sum_{j\in\mathcal{J}} p_j x_{i_1,j} + \frac{1}{2}\sum_{j\in\mathcal{J}} p_j x_{i_2,j} \\ &\leq \frac{u}{2} + \frac{u}{2} = u\end{aligned}$$

and

$$\begin{aligned}\sum_{j\in\mathcal{J}} p_j x'_{i_1,j} = \sum_{j\in\mathcal{J}} p_j x'_{i_2,j} &= \sum_{j\in\mathcal{J}} p_j \frac{x_{i_1,j} + x_{i_2,j}}{2} \\ &= \frac{1}{2}\sum_{j\in\mathcal{J}} p_j x_{i_1,j} + \frac{1}{2}\sum_{j\in\mathcal{J}} p_j x_{i_2,j} \\ &\geq \frac{\ell}{2} + \frac{\ell}{2} = \ell.\end{aligned}$$

Hence, the solution remains optimal. As for the integrality constraints, again the machines $i \notin \{i_1, i_2\}$ do not change. Let $k \in \{1, \ldots, 1/\epsilon\}$. Since $y_{i_1,k} \equiv y_{i_2,k}$, we have that $y_{i_1,k} + y_{i_2,k}$ is even. It follows that

$$\sum_{j\in\mathcal{J}_k} x'_{i_1,j} = \sum_{j\in\mathcal{J}_k} x'_{i_2,j} = \frac{y_{i_1} + y_{i_2}}{2}$$

is integral. Now it remains to show that (2) has decreased. Notice that by triangle inequality

$$\|y'_{i_1,k}\|_2 + \|y'_{i_2,k}\|_2 = 2\left\|\frac{y_{i_1,k} + y_{i_2,k}}{2}\right\|_2 \leq \|y_{i_1,k}\|_2 + \|y_{i_2,k}\|_2$$

and strict inequality holds when $y_{i_1,k} \neq y_{i_2,k}$. Since this is the case for at least one k and all machines $i \notin \{i_1, i_2\}$ do not change, we have that (2) has decreased. A contradiction. ◀

Using Lemma 3 we can solve the slot-MILP in polynomial time.

▶ **Lemma 4.** *We can solve the slot-MILP in time* $m^{O(2^{1/\epsilon})} \cdot n^{O(1/\epsilon \cdot 2^{1/\epsilon})}$.

Proof. Lemma 3 implies that there are only $2^{1/\epsilon}$ many machine types denoted by the $1/\epsilon$-tuples y_i. This allows us to guess all values of $y_{i,k}$ (up to permutations of machines) of the optimal solution as follows. For each of the $2^{1/\epsilon}$ types we guess (1) the number of machines

having this type and (2) for each $k \in \{1, ..., 1/\epsilon\}$ we guess the value of $y_{i,k}$ for each machine $i \in \mathcal{M}$ of this type. Note that the machines are identical and hence it suffices to guess the number of machines of each type, rather than guessing which exact machine is of which type. The total number of guesses is bounded by $m^{O(2^{1/\epsilon})} \cdot n^{O(1/\epsilon \cdot 2^{1/\epsilon})}$. Then the remaining problem is only a linear program since all integral variables of the slot-MILP are already fixed. If our guess was correct then the LP must have a feasible solution. ◀

2.2 Faster (approximate) solution to the relaxation

The solution based on Lemma 3 can be found in double exponential time with respect to the number of job types $1/\epsilon$ and is an exact solution to the slot-MILP. Using a different (slightly more complicated) structural property one can find an additive δ-approximate solution to the slot-MILP in single exponential time with respect to $1/\epsilon$ and polynomial in $1/\delta$, i.e., even with $\delta := 1/n^{O(1)}$ we obtain polynomial running time. Here, δ-approximate means that we find a solution to a weaker version of slot-MILP with lower and upper bounds $\ell' = \ell - \delta \cdot p_{\max}$ and $u' = u + \delta \cdot p_{\max}$ for the load of every machine i. We refer to this weaker MILP the slot-MILP'.

The algorithm is based on a different structural property than the one proved in Lemma 3. Given a solution (x, y) of the slot-MILP, for each machine $i \in \mathcal{M}$ and each $k \in \{1, \dots, 1/\epsilon\}$, we denote by $z_{i,k}$ the average size of the jobs type k on machine i defined by

$$z_{i,k} \cdot y_{i,k} = \sum_{j \in \mathcal{J}_k} p_j x_{i,j}.$$

In the case that $y_{i,k} = 0$ this allows us to freely choose the value of $z_{i,k}$ which is important for the structural property in the following lemma. We prove that there is always a solution to the slot-MILP and an ordering of the machines such that for each $k \in \{1, \dots, 1/\epsilon\}$ the values $z_{i,k}$ are non-decreasing and on each prefix of length ℓ of the machines the total size of the slots for the jobs in $\mathcal{J}_k$ is at least as large as the $y_{\sigma(1),k} + \cdots + y_{\sigma(\ell),k}$ smallest jobs in $\mathcal{J}_k$. For each integer n' let $\mathcal{J}_k^{\min}(n') \subseteq \mathcal{J}_k$ be the n' smallest jobs in $\mathcal{J}_k$.

▶ **Lemma 5.** *There is an optimal solution (x, y) for the slot-MILP, a corresponding vector $\{z_{i,k}\}_{i \in \mathcal{M}, k \in \{1,\dots,1/\epsilon\}}$, and an ordering $\sigma : \{1, ..., |\mathcal{M}|\} \to \mathcal{M}$ such that*

$$\sum_{\ell'=1}^{\ell} y_{\sigma(\ell'),k} z_{\sigma(\ell'),k} \geq \sum_{j \in \mathcal{J}_k^{\min}(y_{\sigma(1),k} + \cdots + y_{\sigma(\ell),k})} p_j \qquad \begin{matrix} \forall k \in \{1, \dots, 1/\epsilon\} \\ \forall \ell \in \{1, ..., |\mathcal{M}|\} \end{matrix} \tag{3}$$

$$\sum_{i \in \mathcal{M}} y_{i,k} z_{i,k} = \sum_{j \in \mathcal{J}_k} p_j \qquad \forall k \in \{1, \dots, 1/\epsilon\} \tag{4}$$

$$z_{\sigma(\ell),k} \leq z_{\sigma(\ell+1),k} \qquad \begin{matrix} \forall k \in \{1, \dots, 1/\epsilon\} \\ \forall \ell \in \{1, ..., |\mathcal{M}| - 1\}. \end{matrix} \tag{5}$$

Proof. Condition (3) and (4) follow directly from feasibility of the solution.

To show condition (5), let x, y be a solution with corresponding average load vector $\{z_{i,k}\}_{i \in \mathcal{M}, k \in 1,\dots,1/\epsilon\}}$, where the values $z_{i,k}$ when $y_{i,k} = 0$ are chosen appropriately. Let $\hat{z}_1 \leq \cdots \leq \hat{z}_{\bar{n}}$ be an ordering of the $\bar{n} = |\{(i,k) : y_{ik} > 0\}|$ values $z_{i,k}$ for all $i \in \mathcal{M}, k \in \{1, \dots, 1/\epsilon\}$ with $y_{ik} > 0$. Assume that (x, y) is the solution maximizing the following potential function

$$\sum_{i=1}^{\bar{n}} n^{2(m/\epsilon - i)} \hat{z}_i. \tag{6}$$

We will now show that in this case we can iteratively find an ordering of machines such that condition (5) holds and otherwise get a contradiction with respect to the potential function. Let i be the machine minimizing $\sum_{k=1}^{1/\epsilon} z_{i,k}$. All other machines i' must satisfy one of the following two cases: (1) $z_{i,k} \leq z_{i',k}$ for all $k \in \{1, ..., 1/\epsilon\}$ or (2) $z_{i,k} > z_{i',k}$ for some k. If (1) holds for all machines i' we relabel machine i as machine 1. Otherwise, let $i' \neq i$ be a machine such that for some k

$$z_{i,k} > z_{i',k}. \tag{7}$$

Then, as i minimizes $\sum_{k=1}^{1/\epsilon} z_{i,k}$ we know that there must exist $\overline{k} \neq k$ with

$$z_{i,\overline{k}} < z_{i',\overline{k}}. \tag{8}$$

As we can freely choose the value of $z_{\overline{i},k'}$, whenever $y_{\overline{i},k'} = 0$, we know that $y_{i,k}, y_{i,\overline{k}}, y_{i',k}, y_{i',\overline{k}} > 0$. We now gradually exchange jobs of $\mathcal{J}_k$ and $\mathcal{J}_{\overline{k}}$ between i and i' without changing the total load on either of the machines. Indeed, there must be some $j, j' \in \mathcal{J}_k$ with $x_{i,j} > 0$, $x_{i',j'} > 0$, and $p_j > p_{j'}$. Conversely, there are $\overline{j}, \overline{j}' \in \mathcal{J}_{\overline{k}}$ with $x_{i,\overline{j}} > 0$, $x_{i',\overline{j}'} > 0$, and $p_{\overline{j}} < p_{\overline{j}'}$. For some $\delta, \overline{\delta} > 0$ we now augment the solution in the following way.

$$\begin{aligned} x_{i,j'} &\leftarrow x_{i,j'} + \delta & \qquad x_{i,\overline{j}'} &\leftarrow x_{i,\overline{j}'} + \overline{\delta} \\ x_{i,j} &\leftarrow x_{i,j} - \delta & \qquad x_{i,\overline{j}} &\leftarrow x_{i,\overline{j}} - \overline{\delta} \\ x_{i',j'} &\leftarrow x_{i',j'} - \delta & \qquad x_{i',\overline{j}'} &\leftarrow x_{i',\overline{j}'} - \overline{\delta} \\ x_{i',j} &\leftarrow x_{i',j} + \delta & \qquad x_{i',\overline{j}} &\leftarrow x_{i',\overline{j}} + \overline{\delta} \end{aligned}$$

It is easy to see that for δ and $\overline{\delta}$ sufficiently small each variable remains non-negative. Moreover, each job remains fully assigned and the number of jobs of $\mathcal{J}_k$ and $\mathcal{J}_{\overline{k}}$ assigned to i and i' remains the same.

By setting $\overline{\delta} = \delta(p_j - p_{j'})/(p_{\overline{j}'} - p_{\overline{j}})$ the load over each of the two machines stays the same. Furthermore, as $p_j > p_{j'}$ and $p_{\overline{j}'} > p_{\overline{j}}$ we have that $\delta, \overline{\delta} > 0$. We choose δ maximal such that all x variables remain non-negative and the inequalities (7) and (8) still hold or turn to equality. This means that we decreased $z_{i,k}$ by $\frac{\delta(p_j - p'_j)}{y_{i,k}}$ and $z_{i',\overline{k}}$ by $\frac{\delta(p_j - p_{j'})}{y_{i,\overline{k}}}$. At the same time we increased $z_{i,\overline{k}}$ by $\frac{\delta(p_j - p_{j'})}{y_{i,\overline{k}}}$ and $z_{i',k}$ by $\frac{\delta(p_j - p'_j)}{y_{i',k}}$. Since $z_{i',k}$ and $z_{i,\bar{k}}$ (the respective smaller z-variables for i and i' that we change) increase by at least $\frac{\delta(p_j - p'_j)}{n}$ and $z_{i,k}$ and $z_{i',\bar{k}}$ decrease by at most $\delta(p_j - p'_j)$, we have that (6) increases. This gives a contradiction.

As we can repeat this argument iteratively assuming that machines $\{1, \ldots, i_0\}$ are correctly sorted for some $i_0 \in \{1, \ldots, m\}$, we have that there exists a solution (x, y) with vector $\{z_{i,k}\}_{i \in \mathcal{M}, k \in 1, \ldots, 1/\epsilon\}}$ such that condition (5) holds. ◀

We introduce a dynamic program that uses the property from Lemma 5. Intuitively, our DP guesses the machines in the ordering σ one after the other. When it guesses the next machine i, it guesses for each $k \in \{1, \ldots, 1/\epsilon\}$ the value $z_{i,k}$ and the number of jobs $y_{i,k}$ from $\mathcal{J}_k$ on machine i. In order to bound the running time we need to consider rounded values of $z_{i,k}$. Therefore, the DP ensures that the conditions (3) and (5) on the vectors y, z from Lemma 5 are satisfied and that condition (4) as well as the upper and lower target load bounds are only violated to a small extent. The following lemma shows that this is sufficient in order to compute an approximate solution to the slot-MILP based on the vectors y, z.

▶ **Lemma 6.** *Suppose that we are given an ordering* $\sigma : \{1, ..., |\mathcal{M}|\} \to \mathcal{M}$ *and vectors* $\{y_{i,k}, z_{i,k}\}_{i\in\mathcal{M},k\in\{1,...,1/\epsilon\}}$ *such that conditions* (3) *and* (5) *hold. Moreover, assume that for each* $i \in \mathcal{M}$ *it holds that*

$$\ell \leq \sum_{k=1}^{1/\epsilon} y_{i,k} z_{i,k} \leq u + \delta p_{\max} \tag{9}$$

and for each $k \in \{1, \cdots, 1/\epsilon\}$ *we have that condition* (4) *is slightly violated as follows*

$$\sum_{j\in\mathcal{J}_k} p_j \leq \sum_{i\in\mathcal{M}} y_{i,k} z_{i,k} \leq \sum_{j\in\mathcal{J}_k} p_j + \delta\epsilon \cdot p_{\max}. \tag{10}$$

Then we can compute a vector $\{x_{i,j}\}_{i\in\mathcal{M},j\in\mathcal{J}}$ *such that* (x, y) *is a solution to slot-MILP' in time* $O(mn^2)$.

Proof. We first show that there exists an assignment vector $\{x_{i,j}\}_{i\in\mathcal{M},j\in\mathcal{J}}$ satisfying

$$\sum_{j\in\mathcal{J}_k} p_j x_{i,j} \leq y_{i,k} z_{i,k} \tag{11}$$

for all $i \in \mathcal{M}$ and $k \in \{1, \ldots, 1/\epsilon\}$. In order to do so we use condition (3) of Lemma 5. We find this assignment independently for all k. We start by assigning $\mathcal{J}_k^{\min}(y_{1,k})$ (completely) to machine 1, then $\mathcal{J}_k^{\min}(y_{1,k} + y_{2,k}) \setminus \mathcal{J}_k^{\min}(y_{1,k})$ to machine 2, etc. This assignment does not necessary have the desired property (11). Hence, we repair the property iteratively for $i = 2, \ldots, m$. Machine 1 clearly satisfies (11) because of (3)). Let $i \in \{2, \ldots, m-1\}$ such that all machines $1, \ldots, i$ satisfy (11). In each iteration i we do not touch any of the machines $i+1, \ldots, m$. Hence, when repairing machine i we may assume that machines $1, \ldots, i$ contain only $\mathcal{J}_k^{\min}(y_{1,k} + \cdots + y_{i,k})$. If machine i satisfies (11) we are done and continue with $i+1$. Otherwise, we know that there is a job j with $p_j > z_{i,k}$ and $x_{i,j} > 0$. Moreover, because of condition (3) we have

$$\sum_{i'=1}^{i} \sum_{j\in\mathcal{J}_k^{\min}(y_{1,k}+\cdots+y_{i,k})} p_j x_{i',j} = \sum_{j\in\mathcal{J}_k^{\min}(y_{1,k}+\cdots+y_{i,k})} p_j \leq y_{1,k} z_{1,k} + \cdots + y_{i,k} z_{i,k}. \tag{12}$$

Since i violates (11) there must be some $i' < i$ satisfying (11) with strict inequality. In particular, there is a job j' with $p_{j'} < z_{i',k} \leq z_{i,k} < p_j$ and $x_{i',j'} > 0$. We now choose an $\alpha > 0$ and exchange j' and j between i and i' as follows

$$x_{i,j'} \leftarrow x_{i,j'} + \alpha \qquad x_{i',j'} \leftarrow x_{i',j'} - \alpha$$
$$x_{i,j} \leftarrow x_{i,j'} - \alpha \qquad x_{i',j'} \leftarrow x_{i',j'} + \alpha$$

Clearly, the solution remains feasible. We choose α maximal such that either i' satisfies (11) with equality, i satisfies (11), $x_{i,j} = 0$, or $x_{i',j'} = 0$. The choice of α makes sure that each pair j, j' that can be exchanged like this will only be exchanged once. This procedure is repeated until i satisfies (11). As the procedure is repeated for all i and possibly has to check all pairs of every job type in each exchange we have a running time of $O(mn^2)$.

Next, we claim that for all i and k, we have that (13) holds, that is,

$$\sum_{j\in\mathcal{J}_k} p_j x_{i,j} \geq y_{i,k} z_{i,k} - \delta\epsilon \cdot p_{\max}. \tag{13}$$

To prove this claim, assume by contradiction that for some machine i' (13) does not hold. Then by (11) and condition (4), we have that

$$\sum_{j\in\mathcal{J}_k}\sum_{i\in\mathcal{M}} x_{i,j}p_j < \sum_{i\in\mathcal{M}} y_{i,k}z_{i,k} - \delta\epsilon\cdot p_{\max} < \sum_{j\in\mathcal{J}_k} p_j. \tag{14}$$

This contradicts the fact that all jobs are fully assigned and thus $\sum_{j\in\mathcal{J}_k} p_j x_{i,j} = \sum_{j\in\mathcal{J}_k} p_j$. Hence, we have that

$$\sum_{k=1}^{1/\epsilon}\sum_{k\in\mathcal{J}_k} p_j x_{ij} \geq \sum_{k=1}^{1/\epsilon} y_{i,k}z_{i,k} - \delta p_{\max} \geq \ell - \delta p_{\max}, \tag{15}$$

where the last inequality follows by condition (9). ◀

The goal of our DP is to compute vectors $\{y_{i,k}, z_{i,k}\}_{i\in\mathcal{M},k\in\{1,\dots,1/\epsilon\}}$ that satisfy the conditions due to Lemma 6. The key insight is now that when we consider the next machine i' in the ordering, we do not need to remember all vectors $\{y_{i,k}, z_{i,k}\}_{i\in\mathcal{M},k\in\{1,\dots,1/\epsilon\}}$ for all previously considered machines i, but it suffices to remember the number of previously assigned jobs from each set $\mathcal{J}_k$, the current left hand side of inequality (3) for each k and the vector $\{z_{i'',k}\}_{k\in\{1,\dots,1/\epsilon\}}$ of the previously considered machine i''. At each iteration the DP then guesses the vectors $\{y_{i,k}, z_{i,k}\}_{i\in\mathcal{M},k\in\{1,\dots,1/\epsilon\}}$ such that the new solution consisting of the guess for machine i and the remembered solution for the previous machines satisfies the conditions stated in Lemma 7. If none of the guesses satisfies these conditions the DP cell corresponding to this iteration remains empty.

To give a more detailed description, we introduce a DP-table with one cell for each combination of

- a value $i \in \{0, ..., m\}$ indicating the number of machines that have already been considered,
- a vector $\{z_{i,k}\}_{k\in\{1,\dots,1/\epsilon\}}$ where $z_{i,k} \in \left\{0, \frac{\delta\epsilon}{n}p_{\max}, \frac{2\delta\epsilon}{n}p_{\max}, ..., p_{\max}\right\}$ for $k \in \{1, \dots, 1/\epsilon\}$. The vector $z_{i,k}$ corresponds to the average loads on the currently considered machine,
- a vector $\{y_{i,k}\}_{k\in\{1,\dots,1/\epsilon\}}$ where $y_{i,k} \in \{0, 1, 2, ..., n_k\}$ for each $k \in \{1, \dots, 1/\epsilon\}$. The vector $y_{i,k}$ corresponds to the number of jobs on the currently considered machine,
- for each $k \in \{1, ..., 1/\epsilon\}$
  - a value $n'_k \in \{0, ..., |\mathcal{J}_k|\}$ indicating the number of slots for jobs of $\mathcal{J}_k$ that have already been assigned to machines,
  - a value S_k with $S_k \in \left\{0, \frac{\delta\epsilon}{n}p_{\max}, \frac{2\delta\epsilon}{n}p_{\max}, ..., np_{\max}\right\}$ which corresponds to the value $\sum_{i'=1}^{i} y_{i',k}z_{i',k}$.

Each cell corresponds to the subproblem of checking whether there is a solution using machines $\{1, \dots, i\}$ such that machine i is assigned $y_{i,k}$ jobs of each type k with average load $z_{i,k}$ and for each type k a total number of n'_k slots is assigned of a total volume of S_k. Due to the dimension of the values corresponding to a DP-cell, the dimension of the DP table is given by $m^2 \cdot (\frac{n}{\delta\epsilon})^{O(1/\epsilon)}$.

When considering cell

$$\left(i, \{z_{i,k}\}_{k\in\{1,\dots,1/\epsilon\}}, \{y_{i,k}\}_{k\in\{1,\dots,1/\epsilon\}}, \{n'_k\}_{k\in\{1,\dots,1/\epsilon\}}, \{S_k\}_{k\in\{1,\dots,1/\epsilon\}}\right)$$

the DP checks whether for some $\{\tilde{z}_{i-1,k}\}_{k\in\{1,\dots,1/\epsilon\}}$ and $\{\tilde{y}_{i-1,k}\}_{k\in\{1,\dots,1/\epsilon\}}$ the entry of the DP is true in cell

$$\left(i-1, \{\tilde{z}_{i-1,k}\}_{k\in\{1,\dots,1/\epsilon\}}, \{\tilde{y}_{i-1,k}\}_{k\in\{1,\dots,1/\epsilon\}}, \{\tilde{n}'_k\}_{k\in\{1,\dots,1/\epsilon\}}, \left\{\tilde{S}_k\right\}_{k\in\{1,\dots,1/\epsilon\}}\right),$$

where $\tilde{n}'_k = n'_k - y_{i,k}$ for each $k \in \{1, \dots, 1/\epsilon\}$, and $\tilde{S}_k = S_k - y_{i,k} z_{i,k}$ for each $k \in \{1, \dots, 1/\epsilon\}$. Then we need to check if the following conditions are true for all $k \in \{1, ..., 1/\epsilon\}$:

$$S_k \leq \delta\epsilon \cdot p_{\max} + \sum_{j \in \mathcal{J}_k} p_j \tag{16}$$

$$z_{i,k} \geq \tilde{z}_{i-1,k}. \tag{17}$$

If these conditions are true, then there exists a solution corresponding to the considered DP cell. Filling each cell takes $(\frac{n}{\delta\epsilon})^{O(1/\epsilon)}$.

Finally, for each possible value of $\{z_{m,k}\}_{k \in \{1,\dots,1/\epsilon\}}$ we check whether there exists a solution for the DP cell

$$\left(m, \{z_{m,k}\}_{k \in \{1,\dots,1/\epsilon\}}, \{y_{m,k}\}_{k \in \{1,\dots,1/\epsilon\}}, \{n'_k\}_{k \in \{1,\dots,1/\epsilon\}}, \{S_k\}_{k \in \{1,\dots,1/\epsilon\}}\right),$$

where all jobs are assigned and for every $k \in \{1, \dots, 1/\epsilon\}$ we have that

$$\sum_{j \in \mathcal{J}_k} p_j \leq S_k \leq \sum_{j \in \mathcal{J}_k} p_j + \delta\epsilon \cdot p_{\max}.$$

If this is the case we use standard backward recursion to find vectors $\{z_{i,k}\}_{k \in \{1,\dots,1/\epsilon\}}$ and $\{y_{i,k}\}_{k \in \{1,\dots,1/\epsilon\}}$ for all $i \in \{1, ..., m\}$ and an assignment of machine types to machine indices and use Lemma 6 to obtain a solution (x, y) to slot-MILP'. If there is no such solution we assert that there is no solution to the original relaxation, i.e., to slot-MILP.

▶ **Lemma 7.** *For each $\delta > 0$ there is an algorithm with a running time of $m^2(\frac{n}{\delta\epsilon})^{O(1/\epsilon)}$ which either finds a δ-approximate solution to the slot-MILP (and thus a feasible solution to slot-MILP') or asserts that the slot-MILP is infeasible.*

Proof. The running time follows from the dimension of the DP table and the time it takes to validate a specific DP cell. This amounts to a running time of $m^2(\frac{n}{\delta\epsilon})^{O(1/\epsilon)}$.

For the correctness of the DP we need two observations: (1) due to conditions (16) and (17) and the way we check whether a solution corresponding to a DP cell exists we have that there exists a solution for machine i if and only if there is a solution for machine $i-1$. Hence, we can indeed find a solution via backward recursion and (2) if for some $\{z_{m,k}\}_{k \in \{1,\dots,1/\epsilon\}}$ and $\{y_{m,k}\}_{k \in \{1,\dots,1/\epsilon\}}$ there is a solution for the cell

$$\left(m, \{z_{m,k}\}_{k \in \{1,\dots,1/\epsilon\}}, \{y_{m,k}\}_{k \in \{1,\dots,1/\epsilon\}}, \{n'_k\}_{k \in \{1,\dots,1/\epsilon\}}, \{S_k\}_{k \in \{1,\dots,1/\epsilon\}}\right),$$

then we can apply Lemma 6 to find a solution to slot-MILP'. If there is no such solution, we know that due to our rounding of the z-values there is also no solution satisfying Lemma 5. This implies that there is no solution to the slot-MILP. ◀

3 Rounding the relaxation

We assume that we are given an exact solution to the slot-MILP via the algorithm due to Lemma 4 or an approximate solution via the algorithm due to Lemma 7. In this section, we describe an algorithm with a running time of $n^{O(1)}$ that computes an integral solution to the slot-MILP (or slot-MILP') which for each machine $i \in \mathcal{M}$ violates the target load intervals by at most $\epsilon \cdot p_{\max}$. For a solution to the slot-MILP this implies that $\sum_{j \in \mathcal{J}} p_j x_{i,j} \in$

$[\ell-\epsilon\cdot p_{\max}, u+\epsilon\cdot p_{\max}]$. For a solution to slot-MILP' this implies that the target load violation is given by the error made due to the approximate solution and due to the rounding, i.e., after rounding the solution it holds that $\sum_{j\in\mathcal{J}} p_j x_{i,j} \in [\ell-\delta\cdot p_{\max}-\epsilon\cdot p_{\max}, u+\delta\cdot p_{\max}+\epsilon\cdot p_{\max}]$. In the following we describe the rounding procedure based on an exact solution to the slot-MILP as the same arguments hold for an approximate solution.

We imagine that each machine $i \in \mathcal{M}$ has $y_{i,k}$ slots for the jobs in $\mathcal{J}_k$, for each $k \in \{1, ..., 1/\epsilon\}$. We say that these slots are of *type* k. Notice that $\sum_{i\in\mathcal{M}} y_{i,k} = |\mathcal{J}_k|$. We compute an initial solution by assigning each job $j \in \mathcal{J}_k$ to an arbitrary slot of type k. In this solution there might be a machine i whose load is not in $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$, i.e., the load is too small or too large. We present a local search algorithm that repeatedly swaps pairs of jobs from the same set $\mathcal{J}_k$ such that eventually each machine $i \in \mathcal{M}$ has a load in $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$ while maintaining the number of jobs from each set $\mathcal{J}_k$ on each machine.

3.1 Local search

We describe how to perform one iteration of the local search algorithm. Each iteration aims at finding a pair of jobs that can be swapped. Let $\mathcal{M}_1$ be the set of machines $i \in \mathcal{M}$ that have a load strictly greater than $u + \epsilon \cdot p_{\max}$. Consider a $k \in \{1, ..., 1/\epsilon\}$ such that a job $j \in \mathcal{J}_k$ is assigned to a machine $i \in \mathcal{M}_1$. We would like to exchange j for a smaller job $j' \in \mathcal{J}_k$ that is assigned to a machine $i' \notin \mathcal{M}_1$. Thus, consider all jobs $j' \in \mathcal{J}_k$ with $p_{j'} < p_j$ which are assigned to a machine $i' \notin \mathcal{M}_1$. If the load of i' is at most u then we exchange j and j' which completes the swap. We try to perform such a swap for each $k \in \{1, ..., 1/\epsilon\}$ such that a job $j \in \mathcal{J}_k$ is assigned to a machine in $\mathcal{M}_1$. If we did not perform a swap then let $\mathcal{M}_2$ denote the set of machines $i' \in \mathcal{M}$ having a job j' which we tried to swap with a job j on a machine $i \in \mathcal{M}_1$, i.e., $\mathcal{M}_2$ contains all machines $i' \in \mathcal{M} \setminus \mathcal{M}_1$ for which there exists a machine $i \in \mathcal{M}_1$ and a $k \in \{1, ..., 1/\epsilon\}$ such that there is a job $j \in \mathcal{J}_k$ assigned to i and a job $j' \in \mathcal{J}_k$ assigned to i' with $p_{j'} < p_j$. Observe that each machine $i' \in \mathcal{M}_2$ has a load of more than u.

Now we repeat this procedure: Suppose that we constructed sets of machines $\mathcal{M}_1, ..., \mathcal{M}_\ell$. For each $k \in \{1, ..., 1/\epsilon\}$ such that there is a job $j \in \mathcal{J}_k$ assigned to a machine $i \in \mathcal{M}_\ell$ consider all jobs $j' \in \mathcal{J}_k$ with $p_{j'} < p_j$, which are assigned to a machine $i' \notin \mathcal{M}_1 \cup \ldots \cup \mathcal{M}_\ell$. If the load on one such machine i' is at most u, then we exchange j and j' which completes the swap. In particular, we do not reuse the constructed sets $\mathcal{M}_1, \ldots, \mathcal{M}_\ell$ for the next swap but we forget these sets before the next swap starts. Otherwise, if each considered machine i' has a load strictly more than u we construct a set $\mathcal{M}_{\ell+1}$ consisting of all these machines i' and continue in the current iteration.

Suppose that at the beginning of a swap there is no machine $i \in \mathcal{M}$ that has a load strictly greater than $u + \epsilon \cdot p_{\max}$. Then, a second stage of the local search algorithm takes place. We take the current solution and perform an analogous procedure in order to ensure that each machine $i \in \mathcal{M}$ has a load of at least $\ell - \epsilon \cdot p_{\max}$. Initially define $\mathcal{M}_1$ to be the set of all machines $i \in \mathcal{M}$ with a load strictly less than $\ell - \epsilon \cdot p_{\max}$. Suppose that we constructed sets of machines $\mathcal{M}_1, ..., \mathcal{M}_\ell$. For each $k \in \{1, ..., 1/\epsilon\}$ such that there is a job $j \in \mathcal{J}_k$ assigned to a machine $i \in \mathcal{M}_\ell$ consider all jobs $j' \in \mathcal{J}_k$ with $p_{j'} > p_j$, which are assigned to a machine $i' \notin \mathcal{M}_1 \cup \ldots \cup \mathcal{M}_\ell$. If the load on one such machine i' is at least ℓ, then we exchange j and j' which completes the swap. Otherwise, if each such machine i' has a load of strictly less than ℓ we construct a set $\mathcal{M}_{\ell+1}$ consisting of all these machines i' and continue. As we never increase the load of a machine with load at least $\ell - \epsilon \cdot p_{\max}$ we do not introduce new violations of the upper bound on the load. The algorithm terminates if the load of each machine $i \in \mathcal{M}$ is within $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$.

3.2 Correctness and running time

We show now that the algorithm terminates in $n^{O(1)}$ time. Then, by construction it outputs a solution in which each machine $i \in \mathcal{M}$ has a load in the interval $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$. In the following we prove this for the first stage of the local search, i.e., when at least one machine has as a load higher than $u + \epsilon p_{\max}$. We first show that in each iteration of the first stage we can find a pair of jobs j, j' to swap. Using a similar argument the same can be shown for the second stage.

▶ **Lemma 8.** *In each iteration the algorithm finds two jobs j, j' that it swaps and finding such a pair can be done in time $O(n^2)$.*

Proof. Suppose towards contradiction that the algorithm does not find two jobs $j, j' \in \mathcal{J}_k$ such that j is assigned to a machine $i \in \mathcal{M}_\ell$ with load more than $u + \epsilon \cdot p_{\max}$ and job j' is assigned to a machine $i' \notin \mathcal{M}_1 \cup \ldots \cup \mathcal{M}_\ell$ and $p'_j < p_j$. This means that the machines in $\mathcal{M}_1 \cup \ldots \cup \mathcal{M}_\ell$ are assigned the smallest jobs of type k while each machine having load at least u. Hence, even a fractional assignment of jobs cannot reduce the total load on these machines. Hence, in a fractional assignment at least one machine must have a load greater than u. This gives a contradiction. When finding a pair of jobs to swap each pair of type k is considered exactly once as each job is assigned fully to a machine. As the existence of a pair was shown for an arbitrary k this gives a worst case running time of $O(n^2)$. ◀

Next, we show that the first stage of the algorithm always terminates after at most $O(n^3)$ swaps. As Lemma 8 states that each swap can be done in time $O(n^2)$, this shows that the first stage finishes in time $n^{O(1)}$. To this end, we give an alternative formulation of the first stage of the algorithm as a repeated breadth-first search (BFS). We construct a weighted, directed graph. It contains one special vertex, the source s, and one vertex for each slot, that is $|\mathcal{J}| + 1$ vertices in total. Each non-source vertex is associated with a machine and a size class. The slots of the same machine form a clique: There is an edge from each slot to the other with weight 0. Furthermore, there is an edge of weight 1 from slot v to w, when (1) v and w are not on the same machine, (2) v and w belong to the same size class, and (3) v is currently assigned a larger job than w. Additionally, there is an edge of weight 0 from the source to every slot on a machine with load more than $u + \epsilon \cdot p_{\max}$. The algorithm performs a BFS on the graph above starting in s. Once it reaches a machine with load at most u, it selects the edge (v, w) over which the machine was reached and swaps the jobs assigned to the slots v and w. This is continued until every machine i is assigned a load at most $u + \epsilon \cdot p_{\max}$. The following lemma shows that the distance between s and other slots does not decrease by a swap.

▶ **Lemma 9.** *The distance from s to any slot does not decrease by a swap in the first stage of the algorithm.*

Proof. Note that when we make a swap, we actually reverse the direction of an edge. To get the main idea of the proof, consider an edge (v, w) that is "swapped". As the search is a BFS, we know that the distance in the original graph to w is equal to the shortest path to v plus 1. When we make the swap, this shortest path is eliminated, whereas all other paths from s to w remain. Therefore, the distance from s to w will not decrease. Furthermore, the distance from s to v did not change due to the swap.

Formally, we observe how the graph changes when swapping two jobs. Throughout the proof the distance $d(w)$ of a slot w is the distance from s to w before the swap is executed. Clearly, removing any edges from the graph cannot decrease the distances of vertices. Edges

between slots of the same machine do not change and no new edges can be added from the source, since during the execution of the algorithm a machine with load at most $u + \epsilon \cdot p_{\max}$ will never exceed $u + \epsilon \cdot p_{\max}$. Hence, it suffices to look at the changes in edges of weight 1. Although such an edge (v, w) might be added to the graph, we will show that this happens only when $d(w) \leq d(v) + 1$. Adding this edges cannot decrease any distances, since the first part of a shortest path using (v, w) could always be replaced by a path to w without this edge.

Now we have to check that these are the only changes made to the graph. Let v, w be the slots in which we exchange the jobs. The size of the job in v decreases; the size of the job in w increases. We only need to look at the incoming and outgoing edges of v and w, since all other edges remain the same.

Consider the incoming edges of v. Since the size of the job in v decreases, there could be new incoming edges. Let (w', v) be an edge of weight 1 that is added. This means the job in slot w' has a larger size than the job on v. Either w' is a slot on the same machine as w or (w', w) is in the graph before the swap. The former case implies that $d(w') = d(w) = d(v)+1$. In the latter case we have $d(v) = d(w) - 1 \leq d(w')$. No outgoing edge from v can be added, since the size of v's job decreases.

Now consider w. Since the size of its job increases, no incoming edge can be added. As for the outgoing edges, let (w, w') be an outgoing edge added by the swap. Then either v and w' are slots on the same machine or (v, w') was is in the graph before the swap. In the former case, $d(w') = d(v) = d(w) - 1$. In the latter case, $d(w') \leq d(v) + 1 = d(w)$. ◀

Using Lemmas 8 and 10 we can bound the maximum number of swaps necessary in the first stage of the algorithm.

▶ **Lemma 10.** *The first stage of the algorithm terminates after at most $O(n^3)$ swaps.*

Proof. Let $j_1, \ldots, j_n$ be the jobs $\mathcal{J}$ in increasing order of size. We claim that the potential

$$\sum_{i=1}^{n} i \cdot d(j_i)$$

increases with every swap. Here $d(j_i)$ denotes the distance from s to the slot to which j_i is assigned. Since the function is integral and bounded by n^3, the claim follows. Let j_k, j_h be the jobs that are swapped. Assume that $k < h$, i.e., $p_{j_k} < p_{j_h}$. Let d, d' be the distance functions before and after the swap. Based on Lemma 9 we have $d'(j_i) \geq d(j_i)$ for all $i \notin \{k, h\}$, $d'(j_h) \geq d(j_k)$ and $d'(j_k) \geq d(j_h)$, since these jobs swapped their slots. It follows that

$$\begin{aligned} k \cdot d'(j_k) + h \cdot d'(j_h) &\geq k \cdot d(j_h) + h \cdot d(j_k) \\ &= k \cdot d(j_h) + (h - k) \cdot \underbrace{d(j_k)}_{> d(j_h)} + k \cdot d(j_k) > h \cdot d(j_h) + k \cdot d(j_k). \end{aligned}$$ ◀

In order to complete the running time analysis we consider the second stage. Again, we construct a graph on $|\mathcal{J}| + 1$ vertices with the difference that we add an edge from s to every slot on a machine with load less than $\ell - \epsilon p_{\max}$ and an edge (v, w) between two slots of the same job type associated to different machines if v is currently assigned a smaller job than w. Following the same ideas as the proofs for the first stage, the running time of the second stage can be shown. For the detailed proofs for the second stage of the algorithm we refer to [6].

▶ **Lemma 11.** *The distance from s to any slot does not decrease by a swap in the second stage of the algorithm.*

▶ **Lemma 12.** *The second stage of the algorithm terminates after at most $O(n^3)$ swaps.*

Together, Lemmas 10 and 12 give the running time of the local search procedure.

▶ **Lemma 13.** *Given a solution to the slot-MILP, in time $n^{O(1)}$ we can compute an integral solution to slot-MILP such that $\sum_{j\in\mathcal{J}} p_j x_{i,j} \in [\ell - \epsilon\cdot p_{\max}, u + \epsilon\cdot p_{\max}]$ for each machine $i \in \mathcal{M}$.*

The main theorem follows from Lemmas 7 and 13.

▶ **Theorem 14.** *There is an algorithm for the target load balancing problem with a running time of $m^2 n^{O(1/\epsilon)}$ that computes a solution in which the load of each machine $i \in \mathcal{M}$ is in $[\ell - \epsilon\cdot p_{\max}, u + \epsilon\cdot p_{\max}]$, or asserts that there is no feasible solution.*

4 Applications

We can use Theorem 14 to obtain additive approximation schemes for *makespan minimization*, the *Santa Claus* problem and the *envy-minimizing Santa Claus* problem on identical machines. The idea is to guess the target load intervals up to multiples of $\epsilon\cdot p_{\max}$ and then applying the algorithm due to Theorem 14.

For $P||C_{\max}$ and the *Santa Claus* problem the guessing procedure can be done in time $O(1/\epsilon)$. For $P||C_{\max}$ we set $\ell = 0$ and guess u as a multiple of $\epsilon\cdot p_{\max}$ within the interval $[\frac{1}{m}\sum_{j=1}^n p_j, \frac{1}{m}\sum_{j=1}^n p_j + p_{\max}]$.

▶ **Corollary 15.** *There is an algorithm for $P||C_{\max}$ with a running time of $m^2 n^{O(1/\epsilon)}$ that computes a solution with makespan at most $OPT + \epsilon\cdot p_{\max}$.*

For the *Santa Claus* problem we set $u = \sum_{j=1}^n p_j$ and guess ℓ within the interval $[\frac{1}{m}\sum_{j=1}^n p_j - p_{\max}, \frac{1}{m}\sum_{j=1}^n p_j]$.

▶ **Corollary 16.** *There is an algorithm for the Santa Claus problem on identical machines with a running time of $m^2 n^{O(1/\epsilon)}$ that computes a solution in which each machine has a load of at least $OPT - \epsilon\cdot p_{\max}$.*

For the *envy-minimizing* Santa Claus problem we need to guess both ℓ and u simultaneously from the intervals above with a total number of $O(1/\epsilon^2)$ guesses.

▶ **Corollary 17.** *There is an algorithm for envy-minimization on identical machines with a running time of $m^2 n^{O(1/\epsilon)}$ which computes a solution with envy at most $OPT + \epsilon\cdot p_{\max}$.*

5 Conclusion

In this paper, we formalized the concept of additive approximation schemes and presented such algorithms for makespan minimization, the Santa Claus problem, and minimizing the maximum envy on identical machines with respect to the parameter $p_{\max}$. For the former two problems, additive approximation schemes are particularly interesting when $p_{\max} \ll \mathrm{OPT}$. For the latter, a multiplicative approximation guarantee is not possible for polynomial time algorithms (unless P=NP); therefore, an additive approximation scheme is a suitable alternative way to obtain non-trivial approximation guarantees.

For $P||C_{\max}$ and the Santa Claus problem on identical machines there is an EPTAS [2,21]. We leave as an open question to find additive approximation schemes for these problems with a running time of this form or to rule out that such schemes exists. Note that using similar techniques as in [34], one can easily show that all three versions admit even an additive FPTAS in case that m is a constant.

References

1 N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1:55–66, 1998.

2 N. Alon, A. Shapira, and B. Sudakov. Additive approximation for edge-deletion problems. *Annals of Mathematics*, 170:37–411, 2009.

3 C. Annamalai, C. Kalaitzis, and O. Svensson. Combinatorial algorithm for restricted max-min fair allocation. *ACM Transactions on Algorithms*, 13(3):37:1–37:28, 2017.

4 A. Asadpour, U. Feige, and A. Saberi. Santa claus meets hypergraph matchings. *ACM Transactions on Algorithms*, 8:24:1–24:9, 2012.

5 N. Bansal and M. Sviridenko. The santa claus problem. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, STOC*, pages 31–40, 2006.

6 M. Buchem, L. Rohwedder, T. Vredeveld, and A. Wiese. Additive approximation schemes for load balancing problems. *CoRR*, abs/2007.09333, 2020. `arXiv:2007.09333`.

7 D. Chakrabarty. Max-min allocation. In M.-Y. Kao, editor, *Encyclopedia of Algorithms*, pages 1244–1247. Springer US, Boston, MA, 2016.

8 L. Chen, K. Jansen, and G. Zhang. On the optimality of exact and approximation algorithms for scheduling problems. *Journal of Computer and System Sciences*, 96:1–32, 2018.

9 S.-W. Cheng and Y. Mao. Restricted max-min allocation: Approximation and integrality gap. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming, ICALP*, pages 38:1–38:13, 2019.

10 S. Davies, T. Rothvoss, and Y. Zhang. A tale of santa claus, hypergraphs and matroids. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2748–2757, 2020.

11 F. Eisenbrand and G. Shmonin. Carathéodory bounds for integer cones. *Operations Research Letters*, 34(5):564–568, 2006.

12 U. Feige. On allocations that maximize fairness. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 287–293. SIAM, 2008.

13 M. R. Garey and D. S. Johnson. "strong" NP-completeness results: motivation, examples, and implications. *Journal of the ACM*, 25:499–508, 1978.

14 R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

15 R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.

16 B. Haeupler, B. Saha, and A. Srinivasan. New constructive aspects of the lovász local lemma. *Journal of the ACM*, 58, 2011.

17 R. Hoberg and T. Rothvoss. A logarithmic additive integrality gap for bin packing. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2616–2625. SIAM, 2017.

18 D. S. Hochbaum. Various notions of approximations: Good, better, best and more. *Approximation algorithms for NP-hard problems*, 1997.

19 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.

20 K. Jansen. An EPTAS for scheduling jobs on uniform processors: Using an MILP relaxation with a constant number of integral variables. *SIAM Journal on Discrete Mathematics*, 24:457–485, 2010.

21 K. Jansen, K.-M. Klein, and J. Verschae. Closing the gap for makespan scheduling via sparsification techniques. *Mathematics of Operations Research*, 45:1371–1392, 2020.

22 K. Jansen, S. Kratsch, D. Marx, and I. Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79:39–49, 2013.

23 K. Jansen and L. Rohwedder. On the configuration-lp of the restricted assignment problem. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2670–2678, 2017.

24 K. Jansen and L. Rohwedder. A quasi-polynomial approximation for the restricted assignment problem. In *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization, IPCO*, pages 305–316, 2017.

25 N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, FOCS*, pages 312–320. IEEE Computer Society, 1982.

26 D. Knop and M. Koutecký. Scheduling meets n-fold integer programming. *Journal of Scheduling*, 21(5):493–503, 2018.

27 D. Knop, M. Koutecký, and M. Mnich. Combinatorial n-fold integer programming and applications. *Mathematical Programming*, 184(1):1–34, 2020.

28 A. Kurpisz, M. Mastrolilli, C. Mathieu, T. Mömke, V. Verdugo, and A. Wiese. Semidefinite and linear programming integrality gaps for scheduling identical machines. *Math. Program.*, 172(1-2):231–248, 2018.

29 J. K. Lenstra, D. B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

30 R. J. Lipton, E. Markakis, E. Mossel, and A. Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the 5th ACM Conference on Electronic Commerce, EC*, pages 125–131, 2004.

31 M. Mnich and A. Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1-2):533–562, 2015.

32 T. Ophelders, R. Lambers, F. C. R. Spieksma, and T. Vredeveld. A note on equitable hamiltonian cycles. *Discrete Applied Mathematics*, page forthcoming, 2021.

33 T. Rothvoss. Better bin packing approximations via discrepancy theory. *SIAM Journal on Computing*, 45(3):930–946, 2016.

34 S. K. Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23(1):116–127, 1976.

35 O. Svensson. Santa claus schedules jobs on unrelated machines. *SIAM Journal on Computing*, 41(5):1318–1341, 2012.

36 V. G. Vizing. On an estimate of the chromatic class of a p-graph (in russian). *Diskret. Analiz*, 3:25–30, 1964.

37 G. J. Woeginger. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Operations Research Letters*, 20:149–154, 1997.