Approximating Bounded Job Start Scheduling with Application in Royal Mail Deliveries Under Uncertainty

Jeremy T. Bradley[1], Dimitrios Letsios[2](✉), Ruth Misener[2], and Natasha Page[2]

[1] GBI/Data Science Group, Royal Mail, London, UK
jeremy.bradley@royalmail.com
[2] Department of Computing, Imperial College London, London, UK
{d.letsios,r.misener,natasha.page17}@imperial.ac.uk

Abstract. Motivated by mail delivery scheduling problems arising in Royal Mail, we study a generalization of the fundamental makespan scheduling problem $P||C_{\max}$ which we call the *Bounded Job Start Scheduling Problem*. Given a set of jobs, each one specified by an integer processing time p_j, that have to be executed non-preemptively by a set of m parallel identical machines, the objective is to compute a minimum makespan schedule subject to an upper bound $g \leq m$ on the number of jobs that may simultaneously begin per unit of time. We show that Longest Processing Time First (LPT) algorithm is tightly 2-approximate. After proving that the problem is strongly $\mathcal{NP}$-hard even when $g = 1$, we elaborate on improving the 2-approximation ratio for this case. We distinguish the classes of long and short instances satisfying $p_j \geq m$ and $p_j < m$, respectively, for each job j. We show that LPT is 5/3-approximate for the former and optimal for the latter. Then, we explore the idea of scheduling long jobs in parallel with short jobs to obtain solutions with tightly satisfied packing and bounded job start constraints. For a broad family of instances excluding degenerate instances with many very long jobs and instances with few machines, we derive a 1.985-approximation ratio. For general instances, we require machine augmentation to obtain better than 2-approximate schedules. Finally, we exploit machine augmentation and a state-of-the-art lexicographic optimization method for $P||C_{\max}$ under uncertainty to propose a two-stage robust optimization approach for bounded job start scheduling under uncertainty attaining good trade-offs in terms of makespan and number of used machines. We substantiate this approach numerically using Royal Mail data.

Keywords: Bounded job start scheduling · Approximation algorithms · Robust scheduling · Mail deliveries

1 Introduction

Royal Mail provides mail collection and delivery services for all United Kingdom (UK) addresses. With a small van fleet of 45,000 vehicles and 90,000 drivers

Y. Li et al. (Eds.): COCOA 2019, LNCS 11949, pp. 69–81, 2019.
https://doi.org/10.1007/978-3-030-36412-0_6

delivering to 27 million locations in UK, efficient resource allocation is essential to guarantee the business viability. The backbone of the Royal Mail distribution network is a three-layer hierarchical network with 6 regional distribution centers serving 38 mail centers. Each mail center receives, processes and distributes mail for a large geographically-defined area via 1,250 delivery offices, each serving disjoint sets of neighboring post codes. Mail is collected in mail centers, sorted by region and forwarded to an appropriate onward mail center, making use of the regional distribution centers for cross-docking purposes. From the onward mail center it is transferred to the final delivery office destination. This process has to be completed within 12 to 16 h for 1st class post and 24 to 36 h for 2nd class post depending on when the initial collection takes place.

In a delivery office, post is sorted, divided into routes and delivered to addresses using the a combination of small fleet vans and walked trolleys. Allocation of delivery itineraries to vans is critical. Each delivery office has an exit gate for vans upper bounding the number of vehicles departing per unit of time. Thus, we deal with the problem of scheduling a set $\mathcal{J}$ of jobs (delivery itineraries) each one associated with an integer processing time p_j, on m parallel identical machines (vehicles), s.t. the makespan, i.e. the last job completion time, is minimized. Parameter g upper bounds the number of jobs that may simultaneously begin per unit of time. Each job has to be executed non-preemptively, i.e. by a single machine in a continuous time interval without interruptions. We call this problem the *Bounded Job Start Scheduling Problem (BJSP).*

BJSP is strongly related to the fundamental makespan scheduling problem $P||C_{\max}$ [5]. BJSP generalizes $P||C_{\max}$ as the problems become equivalent when $g = m$. Furthermore, $P||C_{\max}$ is the BJSP relaxation obtained by dropping the BJSP constraint. Note that the $P||C_{\max}$ optimal solution is a factor $\Omega(m)$ from the BJSP one, in the worst case. For example, take an arbitrary $P||C_{\max}$ instance and construct a BJSP one with $g = 1$ by adding a large number of unit jobs. The BJSP optimal schedule requires time intervals during which $m-1$ machines are idle at each time while the $P||C_{\max}$ optimal schedule is perfectly balanced and all machines are busy until the last job completes. On the positive side, we may convert any ρ-approximation algorithm for $P||C_{\max}$ into 2ρ-approximation algorithm for BJSP using naive bounds. Given that $P||C_{\max}$ admits a PTAS, we obtain an $O(n^{1/\epsilon} \cdot poly(n))$-time $(2+\epsilon)$-approximation algorithm for BJSP. Here, a main goal is to obtain tighter performance guarantees. Similarly to $P||C_{\max}$, provably good BJSP solutions must attain low imbalance $\max_i\{T - T_i\}$, where T and T_i are the makespan and completion time of machine i, respectively. Because of the BJSP constraint, feasible schedules may require idle machine time before all jobs have begun. So, BJSP exhibits the additional difficulty of effectively bounding the total idle period $\sum_{t \leq r}(m - |\mathcal{A}_t|)$, where r and $\mathcal{A}_t$ are the last job start time and set of jobs executed during $[t, t+1)$, respectively.

BJSP relaxes the scheduling problem with forbidden sets, i.e. non-overlapping constraints, where subsets of jobs cannot run in parallel [10]. For the latter problem, better than 2-approximation algorithms are ruled out, unless $\mathcal{P} = \mathcal{NP}$ [10]. Even when there is a strict order between jobs in the same forbidden set, the scheduling with forbidden sets problem is equivalent to the precedence-constrained scheduling problem $P|prec|C_{\max}$ and cannot be approximated by a factor lower than $(2-\epsilon)$, assuming a variant of the unique games conjecture [12]. Also, BJSP relaxes the scheduling with forbidden job start times problem, where no job may begin at certain time points, which does not admit constant-factor approximation algorithms [2,3,8,9]. Despite the commonalities with the aforementioned literature, to the authors' knowledge, there is a lack of approximation algorithms for scheduling problems with bounded job starts.

Contributions and Paper Organization. Section 2 formally defines BJSP, proves the problem's $\mathcal{NP}$-hardness, and derives a $O(\log m)$ integrality gap for a natural integer programming formulation. Section 3 investigates *Longest Processing Time First (LPT)* algorithm, i.e. the probably simplest option for BJSP, and derives a tight 2-approximation ratio. The remainder of the paper elaborates on improving this ratio for the special case $g = 1$. Section 2 shows that BJSP remains strongly $\mathcal{NP}$-hard even in this case. Note that several arguments in our analysis can be extended to the arbitrary g case. Focusing on $g = 1$ allows to avoid many floors, ceilings, and simplifies our presentation. Furthermore, any Royal Mail instance can be converted to this case using small discretization.

Section 4 distinguishes between long and short instances. An instance $\langle m, \mathcal{J}\rangle$ is *long* if $p_j \geq m$ for each $j \in \mathcal{J}$ and *short* if $p_j < m$ for all $j \in \mathcal{J}$. This distinction is motivated by the observation that idle time occurs mainly because of (i) simultaneous job completions in the former case, and (ii) limited allowable parallel job executions in the latter case. Section 4 proves that LPT is 5/3-approximate for long instances and optimal for short instances. A key ingredient for establishing the ratio in the case of long instances is a concave relaxation for bounding the idle machine time. Section 4 also obtains an improved approximation ratio for long instances when the maximum job processing time is relatively small using the *Shortest Processing Time First (SPT)* algorithm.

Greedy scheduling policies, e.g. LPT and SPT, that sequence long jobs first and short jobs next, or vice versa, cannot achieve an approximation ratio better than 2. Section 5 proposes the *Long-Short Mixing (LSM)* algorithm that devotes a certain number of machines to long jobs and uses all remaining machines for short jobs. By executing the two types of jobs in parallel, LSM achieves a 1.985-approximation ratio for a broad family of instances. For degenerate instances with many very long jobs or with few machines, we require constant-factor machine augmentation, i.e. fm machines where $f > 1$ is constant, to achieve a strictly lower than 2 approximation ratio.

Because Royal Mail delivery scheduling is subject to uncertainty, the full paper version exploits machine augmentation and a state-of-the-art lexicographic optimization method for $P||C_{\max}$ under uncertainty [6,11] to construct a two-stage robust optimization approach [1,4,7] for the BJSP under uncertainty. We substantiate the proposed approach empirically using Royal Mail data.

Section 6 concludes with a collection of intriguing future directions. Due to space constraints, omitted proofs and parts of the paper are deferred to the full version.

2 Problem Definition and Preliminary Results

An instance $I = \langle m, \mathcal{J} \rangle$ of the *Bounded Job Start Scheduling Problem (BJSP)* is specified by a set $\mathcal{M} = \{1, \dots, m\}$ of parallel identical machines, and a set $\mathcal{J} = \{1, \dots, n\}$ of jobs. A machine may execute at most one job per unit of time. Job $j \in \mathcal{J}$ is associated with an integer processing time p_j. Time is partitioned into a set $D = \{1, \dots, \tau\}$ of discrete time slots. Time slot $t \in T$ corresponds to time interval $[t-1, t)$. Each job should be executed non-preemptively, i.e. in a continuous time interval without interruptions, by a single machine. This interval should consist of an integer number of time slots. The remainder of the manuscript assumes time intervals $[s, t] = \{s, s+1, \dots, t\}$ of time slots. *BJSP parameter* g imposes an upper bound on the number of jobs that may begin per unit of time. The goal is to assign each job $j \in \mathcal{J}$ to a machine and decide its starting time so that this BJSP constraint is not violated and the makespan, i.e. the time at which the last job completes, is minimized. Consider a feasible schedule $\mathcal{S}$ with makespan T. Denote the start time of job j by s_j. Job $j \in \mathcal{J}$ must be entirely executed during the interval $[s_j, C_j)$, where $C_j = s_j + p_j$ is the completion time of j. So, $T = \max_{j \in \mathcal{J}}\{C_j\}$. We say that job j is *alive* at time slot t if $t \in [s_j, C_j)$. Let $\mathcal{A}_t = \{j : t \in [s_j, C_j)\}$ and $\mathcal{B}_t = \{j : s_j = t - 1\}$ be the set of alive and beginning jobs during time slot t, respectively. Schedule $\mathcal{S}$ is feasible only if $|\mathcal{A}_t| \leq m$ and $|\mathcal{B}_t| \leq g$, for all t.

BJSP is strongly $\mathcal{NP}$-hard because it becomes equivalent with $P||C_{\max}$ in the special case where $g = \min\{m, n\}$. Theorem 1 shows that BJSP is strongly $\mathcal{NP}$-hard also when $g = 1$, through a reduction from 3-Partition.

Theorem 1. *BJSP is strongly $\mathcal{NP}$-hard in the special case $g = 1$.*

Theorem 2 shows that a natural integer programming formulation has non-constant integrality gap. Thus, stronger linear programming (LP) relaxations are required for obtaining constant-factor approximation LP rounding algorithms.

Theorem 2. *A natural integer programming formulation using binary variables indicating the start time of each job has integrality gap $\Omega(\log m)$.*

3 LPT Algorithm

Longest Processing Time first algorithm (LPT) schedules the jobs on a fixed number m of machines w.r.t. the order $p_1 \geq \ldots \geq p_n$. Recall that $|\mathcal{A}_t|$ and $|\mathcal{B}_t|$ is the number of alive and beginning jobs, respectively, at time slot $t \in D$. We say that time slot $t \in D$ is *available* if $|\mathcal{A}_t| < m$ and $|\mathcal{B}_t| < g$. LPT schedules the jobs greedily w.r.t. their sorted order. Each job j is scheduled in the earliest available time slot, i.e. at $s_j = \min\{t : |\mathcal{A}_t| < m, |\mathcal{B}_t| < g, t \in D\}$. Theorem 3 proves a tight approximation ratio of 2 for LPT.

Theorem 3. *LPT is tightly 2-approximate for minimizing makespan.*

Proof. Denote by $\mathcal{S}$ and $\mathcal{S}^*$ the LPT and a minimum makespan schedule, respectively. Let ℓ be the job completing last in $\mathcal{S}$, i.e. $T = s_\ell + p_\ell$. For each time slot $t \leq s_\ell$, either $|\mathcal{A}_t| = m$, or $|\mathcal{A}_t| < m$. Since ℓ is scheduled at the earliest available time slot, for each $t \leq s_\ell$ s.t. $|\mathcal{A}_t| < m$, we have that $|\mathcal{B}_t| = g$. Let λ be the total length of time s.t. $|\mathcal{A}_t| < m$ in $\mathcal{S}$. Because of the BJSP constraint, exactly g jobs begin per unit of time, which implies that $\lambda \leq \lceil \frac{\ell}{g} \rceil$. Therefore, schedule $\mathcal{S}$ has makespan

$$T = s_\ell + p_\ell \leq \frac{1}{m}\sum_{j \neq \ell} p_j + \lambda + p_\ell \leq \frac{1}{m}\sum_{j=1}^{n} p_j + \left(\left\lceil \frac{\ell}{g} \right\rceil + p_\ell \right)$$

Denote by s_j^* the starting time of job j in $\mathcal{S}^*$ and let $\pi_1, \ldots, \pi_n$ the job indices ordered in non-decreasing schedule $\mathcal{S}^*$ starting times, i.e. $s_{\pi_1}^* \leq \ldots \leq s_{\pi_n}^*$. Because of the BJSP constraint, $s_{\pi_j}^* \geq \lceil j/g \rceil$. In addition, there exists $j' \in [j, n]$ s.t. $p_{\pi_{j'}} \geq p_j$. Thus, $\max_{j'=j}^{n}\{s_{\pi_{j'}}^* + p_{\pi_{j'}}\} \geq \lceil j/g \rceil + p_j$, for $j = 1, \ldots, n$. Then,

$$T^* \geq \max\left\{ \frac{1}{m}\sum_{j=1}^{n} p_j, \max_{j=1}^{n}\left\{ \left\lceil \frac{j}{g} \right\rceil + p_j \right\} \right\}$$

We conclude that $T \leq 2T^*$.

For the tightness of the analysis, consider an instance $I = \langle m, \mathcal{J} \rangle$ with $m(m-1)$ long jobs of processing time p, where $p = \omega(m)$ and $m = \omega(1)$, $m(p-m)$ unit jobs, and BJSP parameter $g = 1$. LPT schedules the long jobs into $m-1$ batches, each one with exactly m jobs. All jobs of a batch are executed in parallel for their greatest part. In particular, the i-th job of the k-th batch is executed by machine i starting at time slot $(k-1)p + i$. All unit jobs are executed sequentially by machine 1 starting at $(m-1)p+1$. Observe that $\mathcal{S}$ is feasible and has makespan $T = (m-1)p + m(p-m) = (2m-1)p - m^2$. The optimal solution $\mathcal{S}^*$ schedules all jobs in m batches. The k-the batch contains $(m-1)$ long jobs and $(p-m+1)$ unit jobs. Specifically, the i-th long job is executed by machine i beginning at $(k-1)p+i$, while all short jobs are executed consecutively by machine m starting at $(k-1)p+m$ and completing at kp. Schedule $\mathcal{S}^*$ is feasible and has makespan $T^* = mp$. Because $\frac{m}{p} \to 0$ and $\frac{1}{m} \to 0$, i.e. both approach zero, $T \to 2T^*$. □

4 Long and Short Instances

Next, we consider two natural classes of BJSP instances, in the case where $g = 1$, for which LPT achieves an approximation ratio better than 2. Instance $\langle m, \mathcal{J} \rangle$ is (i) *long* if $p_j \geq m$ for each $j \in \mathcal{J}$, and (ii) *short* if $p_j < m$ for every $j \in \mathcal{J}$. This section proves that LPT is 5/3-approximate for long instances and optimal for short instances. Finally, when the longest job is relatively small compared to the total load, we show a better than 5/3-approximation ratio for SPT in the case of long instances.

Consider a feasible schedule $\mathcal{S}$ and let $r = \max_{j \in \mathcal{J}}\{s_j\}$ be the last job start time. We say that $\mathcal{S}$ is a *compact schedule* if it holds that either (i) $|\mathcal{A}_t| = m$, or (ii) $|\mathcal{B}_t| = 1$, for each $t \in [1, r]$. Lemma 1 shows the existence of an optimal compact schedule and derives a lower bound on the optimal makespan.

Lemma 1. *For each instance $I = \langle m, \mathcal{J} \rangle$, there exists a feasible compact schedule $\mathcal{S}^*$ which is optimal. Let $\mathcal{J}^L = \{j : p_j \geq m, j \in \mathcal{J}\}$. If $|\mathcal{J}^L| \geq m$, then $\mathcal{S}^*$ has makespan $T^* \geq \frac{m-1}{2} + \frac{1}{m}\sum_{j=1}^{n} p_j$.*

Next, we analyze LPT in the case of long instances. Similarly to the Lemma 1 proof, we may show that LPT produces a compact schedule $\mathcal{S}$. So, we may partition the interval $[1, r]$ into a sequence $P_1, \ldots, P_k$ of maximal periods satisfying the following invariant: for each $q \in \{1, \ldots, k\}$, either (i) $|\mathcal{A}_t| < m$ for each $t \in P_q$, or (ii) $|\mathcal{A}_t| = m$ for each $t \in P_q$. That is, there is no pair of time slots $s, t \in P_q$ such that $|\mathcal{A}_s| < m$ and $|\mathcal{A}_t| = m$. We say that P_q is a *slack period* if P_q satisfies (i). Otherwise, we refer to P_q as a *full* period. For a given period P_q of length λ_q, denote by $\Lambda_q = \sum_{t \in P_q}(m - |\mathcal{A}_t|)$ the idle machine time. Note that $\Lambda_q = 0$, for each full period P_q. Lemma 2 upper bounds the total idle machine time of slack periods in the LPT schedule $\mathcal{S}$, except the very last period P_k. In the case where P_k is slack, the length λ_k of P_k is upper bounded by Lemma 3.

Lemma 2. *Let $k' = k - 1$. Consider a long instance $I = \langle m, \mathcal{J} \rangle$, with $|\mathcal{J}| \geq m$, and the LPT schedule $\mathcal{S}$. It holds that (i) $\lambda_q \leq m - 1$, and (ii) $\Lambda_q \leq \frac{\lambda_q(\lambda_q - 1)}{2}$ for each slack period P_q, where $q \in \{1, \ldots, k'\}$. Furthermore, (iii) $\sum_{q=1}^{k'} \Lambda_q \leq \frac{nm}{2}$.*

Proof. For part (i), let $P_q = [s, t]$ be a slack time period in $\mathcal{S}$ and assume for contradiction that $\lambda_q \geq m$, i.e. $t \geq s + m - 1$. Given that $p_j \geq m$ for each $j \in \mathcal{J}$, we have that $\{j : s_j \in [s, s + m - 1], j \in \mathcal{J}\} \subseteq \mathcal{A}_{s+m-1}$. That is, all jobs starting during $[s, s + m - 1]$ are alive at time $s + m - 1$. Since P_q is a slack period, it holds that $a_u < m$, for each $u \in [s, s + m - 1]$. Because $\mathcal{S}$ is compact, it must be the case that $b_u = 1$, i.e. exactly $g = 1$ jobs begin, at each $u \in [s, s + m - 1]$. These observations imply that $|\mathcal{A}_{s+m-1}| \geq m$, which contradicts the fact that P_q is a maximal slack period.

For part (ii), we consider the partitioning $\mathcal{A}_u = \mathcal{A}_u^- + \mathcal{A}_u^+$ for each time slot $u \in P_q = [s, t]$, where $\mathcal{A}_u^-$ and $\mathcal{A}_u^+$ is the set of alive jobs at time u completing inside P_q, i.e. $C_j \in [s, t]$, and after P_q, i.e. $C_j > t$, respectively. Since $\lambda_q \leq m - 1$, every job j beginning during P_q, i.e. $s_j \in [s, t]$, must complete after P_q, i.e. $C_j > t$. We modify schedule $\mathcal{S}$ by removing every occurrence of a job j executed inside P_q such that $C_j \in P_q$. Clearly, in the modified schedule $\mathcal{S}'$ the idle time Λ'_q during P_q may only have increased, i.e. $\Lambda_q \leq \Lambda'_q$. Furthermore, no job j with $s_j \in P_q$ is removed. Because $|\mathcal{B}_u| = 1$ for each $u \in P_q$, we have that $|\mathcal{A}_u| = |\mathcal{A}_{u+1}| - 1$ for $u = s, \ldots, t$. Furthermore, $|\mathcal{A}_{t+1}| = m$. So,

$$\Lambda'_q = \sum_{u=s}^{t}(m - |\mathcal{A}_u|) = \sum_{u=s}^{t}[m - (t - s + 1)] = \sum_{u=1}^{\lambda_q - 1} u = \frac{\lambda_q(\lambda_q - 1)}{2}.$$

Now, we proceed with part (iii). Consider a slack period P_q, for $q \in \{1, \ldots, k'\}$. By part (i), $\lambda_q \leq m - 1$. Because of the BJSP constraint, at most

$g = 1$ jobs begin at each $t \in P_q$ and, thus, $\sum_{q=1}^{k'} \lambda_q \leq n-1$. So, by part (ii), the Eq. (1) concave program upper bounds $\sum_{q=1}^{k'} \Lambda_q$.

$$\max_{\lambda_q} \quad \sum_{q=1}^{k'} \frac{\lambda_q(\lambda_q - 1)}{2} \tag{1a}$$

$$1 \leq \lambda_q \leq m \qquad q \in \{1, \ldots, k'\} \tag{1b}$$

$$\sum_{q=1}^{k'} \lambda_q \leq n \tag{1c}$$

Assume without loss of generality that n/m is integer. If $k' \leq n/m$, by setting $\lambda_q = m$, for $q \in \{1, \ldots, k'\}$, we obtain that $\sum_{q=1}^{k'} \lambda_q(\lambda_q - 1)/2 \leq k'm(m-1)/2 \leq nm/2$. If $k' > n/m$, we argue that the solution $\lambda_q = n/m$, for $q \in \{1, \ldots, n/m\}$, and $\lambda_q = 0$, otherwise, is optimal for the concave program (1). In particular, for any solution λ with $0 < \lambda_q, \lambda_{q'} < m$ such that $q \neq q'$, we may construct a modified solution with higher objective value by applying Jensen's inequality $f(\lambda) + f(\lambda') \leq f(\lambda + \lambda')$ for any $\lambda, \lambda' \in [0, \infty)$, with respect to the single variable, convex function $f(\lambda) = \lambda(\lambda - 1)/2$. If $\lambda_q + \lambda_{q'} \leq m$, we may set $\tilde{\lambda}_q = \lambda_q + \lambda_{q'}$ and $\tilde{\lambda}_{q'} = 0$. Otherwise, $m < \lambda_q + \lambda_{q'} \leq 2m$ and we set $\tilde{\lambda}_q = m$ and $\tilde{\lambda}_{q'} = \lambda_q + \lambda_{q'} - m$. In both cases, $\lambda_{q''} = \tilde{\lambda}_{q''}$, for each $q'' \in \{1, \ldots, k'\} \setminus \{q, q'\}$. Clearly, $\tilde{\lambda}$ attains higher objective value than λ, for concave program (1).

Lemma 3. *Suppose that the last period P_k is slack and let $\mathcal{J}_k$ be the set of jobs beginning during P_k. Then, it holds that $\lambda_k \leq \frac{1}{m} \sum_{j \in \mathcal{J}_k} p_j$.*

Theorem 4. *LPT is 5/3-approximate for long instances.*

Proof. Denote the LPT and optimal schedules by $\mathcal{S}$ and $\mathcal{S}^*$, respectively. Let $\ell \in \mathcal{J}$ be a job completing last in $\mathcal{S}$, i.e. $T = s_\ell + p_\ell$. Recall that LPT sorts the jobs s.t. $p_1 \geq \ldots \geq p_n$. W.l.o.g. we may assume that $\ell = \arg\min_{j \in \mathcal{J}}\{p_j\}$. Indeed, we may discard every job $j > \ell$ and bound the algorithm's performance w.r.t. instance $\tilde{I} = \langle m, \mathcal{J} \setminus \{j : j > \ell, j \in \mathcal{J}\}\rangle$. Let $\tilde{S}$ and $\tilde{S}^*$ be the LPT and an optimal schedule attaining makespan $\tilde{T}$ and $\tilde{T}^*$, respectively, for instance $\tilde{I}$. Showing that $\tilde{T} \leq (5/3)\tilde{T}^*$ is sufficient for our purposes because $T = \tilde{T}$ and $\tilde{T}^* \leq T^*$. We distinguish two cases based on whether $p_n > T^*/3$, or $p_n \leq T^*/3$.

In the former case, we claim that $T \leq (3/2)T^*$. Initially, observe that $n \leq 2m$. Otherwise, there would be a machine $i \in \mathcal{M}$ executing at least $|\mathcal{S}_i^*| \geq 3$ jobs, say $j, j', j'' \in \mathcal{J}$, in $\mathcal{S}^*$. This machine would have last job completion time $T_i^* \geq p_j + p_{j'} + p_{j''} > T^*$, which is a contradiction. If $n \leq m$, LPT has clearly makespan $T = T^*$. So, consider that $n > m$. Then, some machine executes at least two jobs in $\mathcal{S}^*$, i.e. $p_n \leq T^*/2$. To prove our claim, it suffices to show that $s_n \leq T^*$. Let $c = \max_{1 \leq j \leq m}\{C_j\}$ be the time at which the last among the m biggest jobs completes. If $s_n \leq c$, then we have that $s_n \leq \max_{1 \leq j \leq m}\{j + p_j\} \leq T^*$. Otherwise, let $\lambda = s_n - c$. Because $n \leq 2m$, it must be the case that $\lambda \leq m$. Furthermore, $|\mathcal{A}_t| < m$ and, thus, $|\mathcal{B}_t| = 1$, for each $t \in [c+1, s_n - 1]$. That is, exactly one job begins per unit of time during $[c+1, s_n]$. Due to the LPT ordering, these are exactly the jobs $\{n-\lambda, \dots, n\}$. Since $\lambda \le m$ and $p_j \ge m$, at least $m-k$ units of time of job $n-k$ are executed from time s_n and onwards, for $k \in \{1, \dots, \lambda\}$. Thus, the total processing load which executed not earlier than s_n is $\mu \ge \sum_{k=1}^{\lambda}(m-k)$. On the other hand, at most $m-k$ machines are idle at time slot $c+k$, for $k \in \{1, \dots, \lambda\}$. So, the total idle machine time during $[m+1, s_n - 1]$ is $\Lambda \le \sum_{k=1}^{\lambda-1}(m-k)$. We conclude that $\mu \ge \Lambda$ which implies that $s_n \le \frac{m(m-1)}{2} + \frac{1}{m}\sum_{j\in\mathcal{J}} p_j$. By Lemma 1, our claim follows.

Now, consider the case $p_n \le T^*/3$. Then,

$$\begin{aligned} T = s_n + p_n &= \frac{1}{m}\left(\sum_{t=1}^{s_n}|\mathcal{A}_t| + \sum_{t=1}^{s_n}(m - |\mathcal{A}_t|)\right) + p_n \\ &\le \frac{1}{m}\left(\sum_{i=1}^{n} p_i + \sum_{q=1}^{\ell}\Lambda_q\right) + p_n \le \frac{1}{m}\sum_{i=1}^{n} p_i + \frac{n}{2} + p_n \le \frac{5}{3}T^*. \end{aligned}$$

The above inequalities hold by (i) the fact that job n completes last, (ii) the definition of alive jobs, (iii) a simple packing argument with job processing times and machine idle time, (iv) Lemmas 1–3, and (v) the bound $T^* \ge \max\{\frac{1}{m}\sum_{j\in\mathcal{J}} p_j, n + p_n, 3p_n\}$, respectively. □

We complement Theorem 4 with a long instance $I = \langle m, \mathcal{J}\rangle$ example for which LPT is 3/2-approximate and leave closing the gap between the two as an open question. Instance I contains $m+1$ jobs, where $p_j = 2m - j$, for $j \in \{1, \dots, m\}$, and $p_{m+1} = m$. In the LPT schedule $\mathcal{S}$, job j is executed at time $s_j = j$, for $j \in \{1, \dots, m\}$, and $s_{m+1} = 2m - 1$. Hence, $T = 3m - 1$. An optimal schedule $\mathcal{S}^*$ assigns job j to machine $j+1$ at time $s_j = j+1$, for $j \in \{1, \dots, m-1\}$. Moreover, jobs m and $m+1$ are assigned to machine 1 beginning at times $s_m = 1$ and $s_{m+1} = m$, respectively. Clearly, $T^* = 2m$.

Theorem 5 uses a simple argument to show that LPT is optimal for short instances.

Theorem 5. *LPT is optimal for short instances.*

Finally, we investigate the performance of *Long Job Shortest Processing Time First (LSPT).* LSPT creates an ordering of the jobs in which (i) each long job precedes every short job, (ii) long jobs are sorted according to *Shortest Processing Time First (SPT)*, and (iii) short jobs are sorted similarly to LPT. LSPT schedules the jobs greedily, in the same vein with LPT, by using this new order of jobs. For long instances, when the largest processing time $p_{\max}$ is relatively small compared to the average machine load, Theorem 6 shows that LSPT achieves an approximation ratio better than the 5/3. From a worst-case analysis viewpoint, the main difference between LSPT and LPT is that the former requires significantly lower idle machine time until the last job start, but at the price of much higher difference between the last job completion times in different machines.

Theorem 6. *LSPT is 2-approximate for minimizing makespan. For long instances, SPT is* $(1+\min\{1, 1/\alpha\})$*-approximate, where* $\alpha = (\frac{1}{m}\sum_{j\in\mathcal{J}} p_j)/p_{\max}$.

5 Parallelizing Long and Short Jobs

This section proposes *Long Short Job Mixing (LSM)* algorithm which is 1.985-approximate for a broad family of instances, e.g. with at most $\lceil 5m/6 \rceil$ jobs of processing time (i) $p_j > (1-\epsilon)(\frac{1}{m}\sum_{j'} p_{j'})$, or (ii) $p_j > (1-\epsilon)\max_{j'}\{j' + p_{j'}\})$ assuming non-increasing p_j's, for small constant $\epsilon > 0$. For degenerate instances with more than $\lceil 5m/6 \rceil$ jobs having $p_j > T^*/2$, where T^* is the optimal objective value, LSM requires bounded machine augmentation to achieve an approximation ratio lower than 2. Note that there can be at most m such jobs. For simplicity, we also assume that $m = \omega(1)$, but the approximation can be adapted for smaller values of m. However, we require that $m \geq 7$.

Algorithm Description. LSM attempts to construct a feasible schedule in which long jobs are executed in parallel with short jobs. To this end, LSM uses $m^L < m$ machines for executing long jobs. The remaining $m^S = m - m^L$ machines are reserved for performing only short jobs. Carefully selecting m^L allows to obtain a good trade-off in terms of (i) delaying long jobs, and (ii) achieving many short job starts at time slots where many long jobs are executed in parallel. Here, we set $m^L = \lceil 5m/6 \rceil$. Before formally presenting LSM, we slightly modify the notions of long and short jobs. In particular, we set $\mathcal{J}^L = \{j : p_j \geq m^L, j \in \mathcal{J}\}$ and $\mathcal{J}^S = \{j : p_j < m^L, j \in \mathcal{J}\}$, respectively. Both $\mathcal{J}^L$ and $\mathcal{J}^S$ are sorted in non-increasing order of processing times. LSM schedules jobs greedily by traversing time slots in increasing order. Let $\mathcal{A}_t^L$ be the set of alive long jobs at time slot $t \in D$. For $t = 1, \dots, \tau$, LSM proceeds as follows: (i) if $|\mathcal{A}_t^L| < m^L$ and $|\mathcal{J}^L| > 0$, then the next long job begins at t, (ii) else if $|\mathcal{J}^S| > 0$ and $m^L \leq |\mathcal{A}_t| < m$, LSM schedules the next short job to start at t, (iii) otherwise, LSM considers the next time slot. From a complementary viewpoint, the set $\mathcal{M}$ of machines is partitioned into $\mathcal{M}^L = \{i : i \leq m^L, i \in \mathcal{M}\}$ and $\mathcal{M}^S = \{i > m^L, i \in \mathcal{M}\}$. LSM prioritizes long jobs on machines $\mathcal{M}^L$ and assigns only short jobs to machines $\mathcal{M}^S$. A job may undergo processing on machine $i \in \mathcal{M}^S$ only if all machines in $\mathcal{M}^L$ are busy. Theorem 7 analyzes LSM.

Theorem 7. *LSM is 1.985-approximate (i) for instances with no more than* $\lceil 5m/6 \rceil$ *jobs s.t.* $p_j > (1-\epsilon)\max\{\frac{1}{m}\sum_{j'} p_{j'}, \max_{j'}\{j' + p_{j'}\}\}$ *for sufficiently large* $\epsilon > 0$, *and (ii) for general instances with 1.2-machine augmentation.*

Proof. Let $\mathcal{S}$ be the LSM schedule and $\ell = \arg\max\{C_j : j \in \mathcal{J}\}$ the job completing last. That is, $\mathcal{S}$ has makespan $T = C_\ell$. For notational convenience, given a subset $P \subseteq D$ of time slots, we denote by $\lambda(P) = |P|$ and $\mu(P) = \sum_{t \in P} |\mathcal{A}_t|$ the number of time slots and executed processing load, respectively, during P. Furthermore, let $n^L = |\mathcal{J}^L|$ and $n^S = |\mathcal{J}^S|$ be the number of long and short jobs, respectively. We distinguish two cases: (i) $\ell \in \mathcal{J}^S$, or (ii) $\ell \in \mathcal{J}^L$.

Case $\ell \in \mathcal{J}^S$. We partition time slots $\{1, \dots, T\}$ into five subsets. Let $r^L = \max_{j \in \mathcal{J}^L}\{s_j\}$ and $r^S = \max_{j \in \mathcal{J}^S}\{s_j\}$ be the maximum long and short job start time, respectively, in $\mathcal{S}$. Since $\ell \in \mathcal{J}^S$, it holds that $r^L < r^S$. For each time slot $t \in [1, r^L]$ in $\mathcal{S}$, either $|\mathcal{A}_t^L| = m^L$ long jobs simultaneously run at t, or not. In the latter case, it must be the case that $t = s_j$ for some $j \in \mathcal{J}^L$. On the other hand, for each time slot $t \in [r^L + 1, s_\ell]$, either $|\mathcal{A}_t| = m$, or $t = s_j$ for some $j \in \mathcal{J}^S$. Finally, $[s_\ell, T(\mathcal{S})]$ is exactly the interval during which job ℓ is executed. If $F^L = \{t : |\mathcal{A}_t^L| = m^L\}$, $B^L = \{t : |\mathcal{A}_t^L| < m^L, t = s_j, j \in \mathcal{J}^L\}$, $F^S = \{t : t > r^L, |\mathcal{A}_t| = m\}$, $B^S = \{t : t > r^L, |\mathcal{A}_t| < m, t = s_j, j \in \mathcal{J}^S\}$, then

$$T \leq \lambda(F^L) + \lambda(B^L) + \lambda(F^S) + \lambda(B^S) + p_\ell. \tag{2}$$

Next, we upper bound a linear combination of $\lambda(F^L)$, $\lambda(B^S)$, and $\lambda(F^S)$ taking into account the fact that certain short jobs begin during a subset $\hat{B}^S \subseteq F^L \cup F^S$ of time slots. By definition, $\lambda(B^S) \leq n^S - \lambda(\hat{B}^S)$. We claim that $\lambda(\hat{B}^S) \geq (m^S/m^L)(\lambda(F^L) + \lambda(F^S))$. For this, consider the time slots $F^L \cup F^S$ as a continuous time period by disregarding intermediate B^L and B^S time slots. Partition this $F^L \cup F^S$ time period into subperiods of equal length m^L. Note that no long job begins during $F^L \cup F^S$ and the machines in $\mathcal{M}^S$ may only execute small jobs in $\mathcal{S}$. Because of the greedy nature of LSM and the fact that $p_j < m^L$ for $j \in \mathcal{J}^S$, there are at least m^S short job starts in each subperiod. Hence, our claim is true and we obtain that $\lambda(B^S) \leq n^S - (m^S/m^L)(\lambda(F^L) + \lambda(F^S))$, or

$$m^S \lambda(F^L) + m^S \lambda(F^S) + m^L \lambda(B^S) \leq m^L n^S. \tag{3}$$

Subsequently, we upper bound a linear combination of $\lambda(F^L)$, $\lambda(B^L)$, and $\lambda(F^S)$ using a simple packing argument. The part of the LSM schedule for long jobs is exactly the LPT schedule for a long instance with n^L jobs and m^L machines. If $|\mathcal{A}_{r^L}^L| < m$, we make the convention that $\mu(B^L)$ does not contain any load of jobs beginning in the maximal slack period completed at time r^L. Observe that $\mu(F^L) = m^L \lambda(F^L)$ and $\mu(F^S) = m\lambda(F^S)$. Additionally, by Lemma 2, we get that $\mu(B^L) \geq m^L \lambda(B^L)/2$, except possibly the very last slack period. Then, by Lemma 3, $\mu(F^L) + \mu(B^L) + \mu(F^S) \leq \sum_{j \in J} p_j$. Hence, we obtain that

$$m^L \lambda(F^L) + \frac{1}{2} m^L \lambda(B^L) + m\lambda(F^S) \leq \sum_{j \in \mathcal{J}} p_j. \tag{4}$$

By summing expressions (3) and (4),

$$(m^L + m^S)\lambda(F^L) + \frac{1}{2} m^L \lambda(B^L) + (m + m^S)\lambda(F^S) + m^L \lambda(B^S) \leq \sum_{j \in \mathcal{J}} p_j + m^L n^S.$$

Because $m = m^L + m^S$, if we divide by m, the last expression gives

$$\lambda(F^L) + \frac{1}{2}(\frac{m^L}{m})\lambda(B^L) + \lambda(F^S) + (\frac{m^L}{m})\lambda(B^S) \leq \frac{1}{m}\sum_{j \in \mathcal{J}} p_j + (\frac{m^L}{m})n^S \tag{5}$$

Now, we distinguish two subcases based on whether $\lambda(F^S) + \lambda(F^L) \geq 5n^S/6$ or not. Obviously, $\lambda(B^L) \leq n^L$. In the former subcase, inequality (3) gives that $\lambda(B^S) \leq (1 - \frac{5m^S}{6m^L})n^S$. Hence, using inequality (5), expression (2) becomes

$$T \leq \frac{1}{m}\sum_{j\in\mathcal{J}} p_j + (1-\frac{m^L}{2m})n^L + \left[(\frac{m^L}{m}) + (1-\frac{m^L}{m})(1-\frac{5m^S}{6m^L})\right] n^S + p_\ell$$

For $m^L = \lceil 5m/6 \rceil$, we have that (i) $5/6 \leq m^L/m \leq 5/6+1/m$, and (ii) $m^S/m^L \geq \frac{1/6-1/m}{5/6+1/m}$. Given that $m = \omega(1)$,

$$T \leq \frac{1}{m}\sum_{j\in\mathcal{J}} p_j + (1-\frac{5}{12})n^L + \left[\frac{5}{6} + (1-\frac{5}{6})(1-\frac{1}{5})\right] n^S + p_\ell$$

Note that an optimal solution $\mathcal{S}^*$ has makespan

$$T^* \geq \max\left\{\frac{1}{m}\sum_{j\in\mathcal{J}} p_j, n^L + n^S + p_\ell\right\}$$

Because the instance has both long and short jobs and $\ell \in \mathcal{J}^S$, it holds that $p_\ell \geq T^*/2$. Therefore, $T \leq (1 + \frac{29}{30} + (\frac{29}{30})\frac{1}{2}) \leq 1.985T^*$. Now, consider the opposite subcase where $\lambda(F^L) + \lambda(F^S) \leq 5n^S/6$. Given that $\lambda(B^L) \leq n^L$ and $\lambda(B^S) \leq n^S$, expression (2) becomes $T \leq \frac{11}{6}(n^S + n^L + p_\ell) \leq 1.835 \cdot T^*$.

Case $\ell \in \mathcal{J}^L$. Recall that $\mathcal{A}_t^L$ and $\mathcal{B}_t^L$ are the sets of long jobs which are alive and begin, respectively, at time slot t. Furthermore, $r^L = \max\{s_j : j \in \mathcal{J}^L\}$ is the last long job starting time. Because LSM greedily uses m^L machines for long jobs, either $|\mathcal{A}_t^L| = m^L$, or $|\mathcal{B}_t^L| = 1$, for each $t \in [1, r^L]$. So, we may partition time slots $\{1, \ldots, r^L\}$ into $F^L = \{t : |\mathcal{A}_t^L| = m\}$ and $B^L = \{t : |\mathcal{A}_t^L| < m, |\mathcal{B}_t^L| = 1\}$ and obtain $T \leq \lambda(F^L) + \lambda(B^L) + p_\ell$. Because m^L long jobs are executed at each time slot $t \in F^L$,

$$\lambda(F^L) \leq \frac{1}{m^L}\left[\sum_{j\in\mathcal{J}^L} p_j - \mu(B^L)\right].$$

Then, Lemma 2 implies that $\mu(B^L) \geq n^L m^L/2$. Furthermore, $\lambda(B^L) \leq n^L$. Therefore, by considering Lemma 3, we obtain

$$T \leq \frac{m}{m^L}\left(\frac{1}{m}\sum_{j\in\mathcal{J}} p_j\right) + \frac{1}{2}(n^L + p_\ell) + \frac{1}{2}p_\ell$$

In the case $p_\ell \leq T^*/2$, since $T^* \geq n^L + p_\ell$, we obtain an approximation ratio of $(\frac{m}{m^L} + \frac{3}{4}) \leq 1.95$, when $m^L = \lceil 5m/6 \rceil$, given that $m = \omega(1)$.

Next, consider the case $p_\ell > T^*/2$. Let $\mathcal{J}^V = \{j : p_j > T^*/2\}$ be the set of very long jobs and $n^V = |\mathcal{J}^V|$. Clearly, $n^V \leq m$. By using resource augmentation, i.e. allowing LSM to use $m' = \lceil 6m/5 \rceil$ machines, we guarantee that LSM assigns at most one job $j \in \mathcal{J}^V$ in each machine. The theorem follows.

Remark. If $\lceil 5m/6 \rceil < |\mathcal{J}^V| \leq m$, LSM is not better than 2-approximate. This can be illustrated with a simple instance consisting of only J^V jobs. Assigning two such jobs on the same machine is pathological. Thus, better than 2-approximate schedules require assigning all jobs in $\mathcal{J}^V$ to different machines.

6 Conclusion

We conclude with a collection of future directions. Because BJSP relaxes scheduling problems with non-overlapping constraints, which do not admit better than 2-approximation algorithms, the existence of such an algorithm without resource augmentation for BJSP is an intriguing open question. A positive answer combining LSM with a new algorithm for instances with many very long jobs is possible. Analyzing the price of robustness of the proposed robust scheduling approach may reveal new insights for effectively solving BJSP under uncertainty. Also, machine augmentation enables more efficient BJSP solving. Integrating multiple delivery offices in a unified setting and performing vehicle sharing on a daily basis enables a practical implementation of machine augmentation when mail delivery demand fluctuates.

References

1. Bertsimas, D., Sim, M.: The price of robustness. Oper. Res. **52**(1), 35–53 (2004)
2. Billaut, J.-C., Sourd, F.: Single machine scheduling with forbidden start times. 4OR **7**(1), 37–50 (2009)
3. Gabay, M., Rapine, C., Brauner, N.: High-multiplicity scheduling on one machine with forbidden start and completion times. J. Sched. **19**(5), 609–616 (2016)
4. Goerigk, M., Schöbel, A.: Algorithm engineering in robust optimization. In: Kliemann, L., Sanders, P. (eds.) Algorithm Engineering. LNCS, vol. 9220, pp. 245–279. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49487-6_8
5. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. **17**(2), 416–429 (1969)
6. Letsios, D., Ruth, M.: Exact lexicographic scheduling and approximate rescheduling. arXiv 1805.03437 (2018)
7. Liebchen, C., Lübbecke, M., Möhring, R., Stiller, S.: The concept of recoverable robustness, linear programming recovery, and railway applications. In: Ahuja, R.K., Möhring, R.H., Zaroliagis, C.D. (eds.) Robust and Online Large-Scale Optimization. LNCS, vol. 5868, pp. 1–27. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-05465-5_1
8. Mnich, M., Bevern, R.V.: Parameterized complexity of machine scheduling: 15 open problems. Comput. Oper. Res. **100**, 254–261 (2018)
9. Rapine, C., Brauner, N.: A polynomial time algorithm for makespan minimization on one machine with forbidden start and completion times. Discrete Optim. **10**(4), 241–250 (2013)

10. Schäffter, M.W.: Scheduling with forbidden sets. Discrete Appl. Math. **72**(1–2), 155–166 (1997)
11. Skutella, M., Verschae, J.: Robust polynomial-time approximation schemes for parallel machine scheduling with job arrivals and departures. Math. Oper. Res. **41**(3), 991–1021 (2016)
12. Svensson, O.: Hardness of precedence constrained scheduling on identical machines. SIAM J. Comput. **40**(5), 1258–1274 (2011)