European Journal of Operational Research 274 (2019) 900–908

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Scheduling with time-of-use costs

Bo Chen [a,*], Xiandong Zhang [b]

[a] Warwick Business School, University of Warwick, Coventry CV4 7AL, UK
[b] School of Management, Fudan University, Shanghai 200433, China

ARTICLE INFO

Article history:
Received 18 December 2017
Accepted 3 November 2018
Available online 15 November 2018

Keywords:
Scheduling
Time-of-use cost
Computational complexity
Efficient algorithm
Polynomial-time approximation scheme

ABSTRACT

We address a class of problems of scheduling a set of independent jobs onto a single machine that charges each job for its processing under variable time-of-use tariffs. Our scheduling objective is to minimize the total cost of processing all the jobs subject to a minimum level of performance in one of the regular scheduling criteria. For each of the problems in the class, we establish its computational tractability. For those that are tractable we provide an efficient algorithm, while for those that are intractable we provide a pseudo-polynomial-time algorithm or a polynomial-time approximation scheme.

1. Introduction

Time-of-use (ToU) tariffs on retail electricity prices have been widely used to manage the balance between supply and demand and to improve the reliability and efficiency of electrical power grids, thanks to the growing infrastructure of advanced metering and monitoring (Braithwait, Hansen, & O'Sheasy, 2007; Fang, Uhan, Zhao, & Sutherland, 2016). For example, as one of the largest combined natural gas and electric energy companies in the United States, PG&E offers ToU plans where they charge higher prices for peak hours and considerably lower prices otherwise (PG&E, 2018). As the demand peaks, the cost of electricity goes up dramatically, due partially to suppliers' reliance on expensive and non-renewable resources like coal to meet the demand (EIA, 2018). Fig. 1 shows a typical variable ToU tariff scheme.

ToU pricing as a technique for matching supply and demand of temporal resources (such as energy, computing resources on a cloud computing platform, and charging stations for electric vehicles) can be seen widely (Chawla et al., 2017).

Wan and Qi (2010) investigate the problem of allocating a single resource of ToU costs to a number of activities (known as *jobs*) to minimize the total resource cost plus a traditional regular scheduling objective, such as total completion time and maximum lateness of jobs. Kulkarni and Munagala (2013) study dynamic variants of such problems in which jobs are released over time and scheduling decisions have to be made online. Fang et al. (2016) focus on minimizing the total cost of resource usage while allowing each job to have a resource demand *in addition to* its processing workload. In this paper, we are concerned with a class of single-resource scheduling problems that are very different from those in the aforementioned studies. In our problems, minimizing the total cost of resource usage is the primary objective, while traditional scheduling objectives are secondary. Research on scheduling under the latter objectives has been extensive and relatively established (see, for example, Chen, Potts, & Woeginger, 1998; Leung, 2004). However, research on scheduling under the former objective is much more scarce, which is, therefore, our focus in this paper. We do not allow offset to each other between the two levels of objectives.

The two-level scheduling problems have been studied in the literature when the objectives of both levels are traditional scheduling objectives. For example, Allahverdi and Aydilek (2014) and Huo and Zhao (2015) consider scheduling problems of minimizing total job completion time subject to the constraint of bounded makespan. These studies can be incorporated into the efforts to tackle more general problems of multiple traditional scheduling objectives, as reviewed by Hoogeveen (2005). According to Hoogeveen (2005), there are four types of approaches to bi-criteria scheduling problems.

- *Lexicographical optimization*: Optimize one objective subject to the constraint that the solution value of another objective is optimized. For example, Gupta and Ruiz-Torres (2000) consider the identical parallel-machine scheduling problem of minimizing makespan subject to minimum total flow-time;

* Corresponding author.
E-mail addresses: b.chen@warwick.ac.uk, bo.chen@wbs.ac.uk (B. Chen), xiandongzhang@fudan.edu.cn (X. Zhang).

https://doi.org/10.1016/j.ejor.2018.11.002
0377-2217/

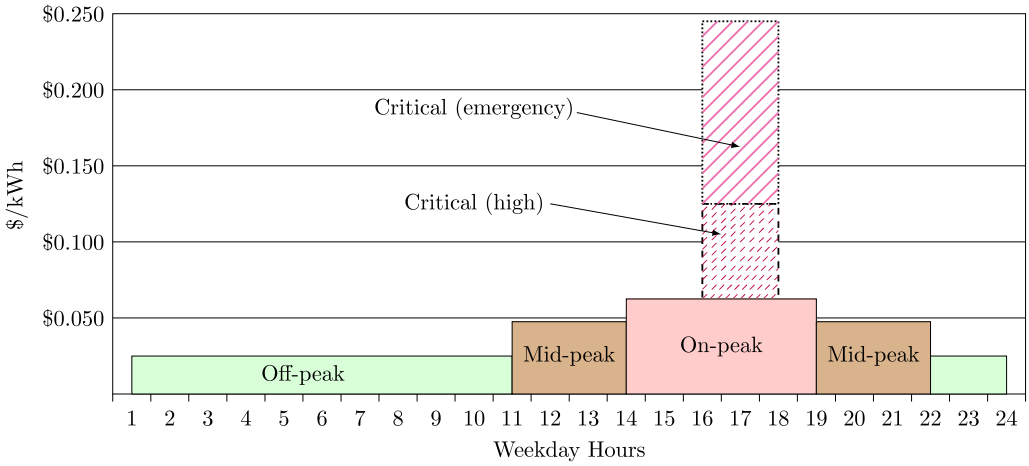

Fig. 1. A variable TOU tariff scheme (Braithwait et al., 2007).

Wan and Yen (2009) consider single-machine scheduling problem of minimizing the total weighted earliness subject to minimum number of tardy jobs.

- *Linear composition objective function*: Optimize a linear composite function of both objectives. Wan and Qi (2010) and Fang et al. (2016) use this type of approach, while Chen and Bulfin (1993) use weighted combinations of maximal tardiness, flow-time and number of tardy jobs.
- *General composition objective function*: Minimize a composite two-objective function that is nondecreasing in both arguments. For example, Hoogeveen and de Velde (1995) apply this approach to the bi-criteria single-machine scheduling problem of minimizing total job completion time and maximum cost simultaneously.
- *Pareto optimization*: Identify all Pareto optimal solutions by solving a series of problems, each of which optimizes one objective given an upper or low bound on another objective. For example, Geng and Yuan (2015) present a polynomial-time algorithm for finding all Pareto optimal solutions in scheduling a batch-machine to minimize the maximum lateness and makespan simultaneously.

In this paper, we study how to schedule a given set of jobs onto a machine with ToU processing costs, so that the total processing cost is minimized subject to that certain scheduling feasibility condition (such as deadline) is satisfied. We refer to this problem as *scheduling with ToU costs* (STOUC). Clearly, the two-level approach we take in this paper is aimed at finding a Pareto optimal solution of the STOUC problem.

Our study on the STOUC problem has been initially motivated by its diverse applications, such as in dynamic pricing for perishable assets (Bitran & Caldentey, 2003; Weatherford & Bodily, 1992). In many situations, decision makers have to consider the changing costs of resources when scheduling their operational activities. Because of the ToU tariffs on electricity prices, manufacturers or service providers can save utility expenses by scheduling their electricity-consuming activities carefully. Due to varying cargo rates and shipping rates (Kasilingam, 1997; Wang, Meng, & Du, 2015), shipping agents of air cargos or liner containers can save logistics cost while satisfying their customer demands. Because prices of passenger airline tickets vary over time (McGill & van Ryzin, 1999), and hotel room rates also change seasonally or have weekly patterns (Badinelli, 2000), travel agencies consider these price fluctuations when designing vacation tour packages, and training organizations adjust their training schedules to save accommodation and travelling costs of their training programs. In health-care sector, physicians and nurses have normal working hours and are costly to work overtime. Appointment scheduling systems need to consider both patient satisfaction and welfare of physicians and nurses (Gupta & Denton, 2008). In a world of smart-metering and other digitized accounting systems, demands for tackling varying resource cost structures are growing.

Our main contributions in this paper are threefold: (a) we answer the question of polynomial-time solvability of the STOUC problem. We show that even the basic STOUC problem, one with a common job deadline, is strongly NP-hard and even with very restricted ToU costs (the cost vector has two "valleys", whose precise definition will follow), it is NP-hard at least in the ordinary sense and is not approximable within any constant factor. We also provide a pseudo-polynomial-time algorithm for the basic STOUC problem with a two-valley ToU cost vector. (b) We solve the basic STOUC problem to optimality if its ToU cost vector has no more than one valley and provide an FPTAS for the basic problem if its ToU cost vector has at most two valleys and the ratios of these costs are bounded above. (c) We establish that, as long as the ToU cost vector has no more than one valley, the STOUC problem with either bounded lateness, or bounded tardiness, or bounded flow-time is polynomially solvable, and the STOUC problem with bounded total of job completion times is also polynomially solvable if additionally the dimension of the cost vector is bounded.

The remaining part of this paper is organized as follows. In Section 2, we give a formal definition of our main problem. In Section 3 we tackle the basic model. We identify precise boundary between tractability and intractability, and provide efficient algorithms and an FPTAS for tractable problems. In Section 5, we extend our basic model to other common scheduling constraints. We provide our conclusions in the last section.

2. Preliminaries and problem complexity

A set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of n independent jobs needs to be scheduled onto a single machine, which is continuously available during the time horizon $[0, T]$ for a given T. Job $J_j \in \mathcal{J}$ requires processing on the single machine for an uninterrupted p_j time units. The time horizon is partitioned into $m \geq 2$ periods $\mathcal{P} = \{\mathcal{P}_1 = [a_0, a_1), \ldots, \mathcal{P}_m = [a_{m-1}, a_m)\}$ and $\Delta_i = a_i - a_{i-1} > 0$ is the duration of period $\mathcal{P}_i$ for $i = 1, \ldots, m$, where $a_m = T$. A unit cost $c_i \geq 0$, which we call *ToU cost*, is incurred if certain amount of time in period $\mathcal{P}_i$ is used for processing a job by the single machine. Denote the sequence of ToU costs by $\pi = (c_1, \ldots, c_m)$. Note that by construction, we have $c_i \neq c_{i+1}$ for $i = 1, \ldots, m-1$. Denote by

$c(t) \in \{c_1, \ldots, c_m\}$ the ToU cost for processing over time interval $[t-1, t)$ for any integer $t \in [0, T]$.

If all jobs are not available for processing at the same time, we use r_j to denote the *release time* of job $J_j \in \mathcal{J}$. Similarly, if individual due time are specified for job completions, we let d_j denote the due time of job $J_j \in \mathcal{J}$.

For ease of presentation, we call a period $\mathcal{P}_i$ a *valley* (respectively, *hill*) if its ToU cost c_i is smaller (respectively, larger) than the ToU cost(s) of its neighboring period(s). For example, in the typical ToU tariffs illustrated in Fig. 1, there are two valleys and one hill. We assume all problem data are integers. Denote

$$P_{st} = \sum_{j=s}^{t} p_j, \quad 1 \le s, t \le n. \tag{1}$$

In particular, denote $P = P_{1n}$ and $P_{st} = 0$ if $s > t$.

In our basic model, we require that all jobs must be completed by a common deadline $\bar{D}$, which is part of the problem specification. In other words, any feasible schedule σ has to satisfy $C_j(\sigma) \le \bar{D}$ for all $J_j \in \mathcal{J}$, where $C_j(\sigma)$ denotes the completion time of job J_j in schedule σ. Without loss of generality, we assume that $\sum_{j=1}^{n} p_j \le \bar{D} = T$. Therefore, in our basic model, we seek a feasible schedule of minimum total execution cost. We refer to the STOUC problem with deadline as the basic STOUC problem. It turns out that our basic STOUC problem is already computationally intractable and difficult to be approximated, as the following theorem states.

Theorem 1. *The basic STOUC problem is strongly NP-hard and, furthermore, not in APX, i.e., there is no polynomial-time ρ-approximation algorithm for any constant $\rho > 1$ unless $P = NP$.*

Proof. We reduce any instance $\mathcal{I}_1$ of the 3-partition problem to an instance $\mathcal{I}_2$ of the basic STOUC problem. Instance $\mathcal{I}_1$: given a positive integer B and a set of $3k$ other positive integers $\{b_1, \ldots, b_{3k}\}$ such that $B/4 < b_j < B/2$ for all $j \in N = \{1, \ldots, 3k\}$. The question: does there exist a partition of the index set N into k 3-element subsets $N_1, \ldots, N_k$ such that $\sum_{j \in N_i} b_j = B$ for all $i = 1, \ldots, k$?

From $\mathcal{I}_1$ we construct instance $\mathcal{I}_2$ of our problem as follows. Let the deadline $\bar{D} = kB + k - 1$ and the number of periods $m = 2k - 1$ with durations and the corresponding ToU costs as follows:

$$\begin{array}{llll} \Delta_{2i-1} = B, & c_{2i-1} = 0, & \text{for } i = 1, \ldots, k; \\ \Delta_{2i} = 1, & c_{2i} = 1, & \text{for } i = 1, \ldots, k-1. \end{array}$$

Then it is clear that the answer to the question for instance $\mathcal{I}_1$ is "yes" if and only if the optimal solution value to the problem instance $\mathcal{I}_2$ is zero, which implies that no constant approximation exists. □

Evidence of intractability of our basic STOUC problem is further strengthened by the following theorem.

Theorem 2. *If π has more than one valley, then the basic STOUC problem is at least ordinary NP-hard and not in APX.*

Proof. We reduce any instance $\mathcal{I}_1$ of the partition problem to an instance $\mathcal{I}_2$ of the basic STOUC problem. Instance $\mathcal{I}_1$: given a positive integer B and a set of k other positive integers $\{b_1, \ldots, b_k\}$, such that $2B = \sum_{j \in N} b_j$, where $N = \{1, \ldots, k\}$, is there a subset N_1 of the index set N such that $\sum_{j \in N_1} b_j = B$?

From $\mathcal{I}_1$ we construct instance $\mathcal{I}_2$ of our problem as follows. Let the deadline $\bar{D} = 2B + 1$ and the number of periods $m = 3$ with durations and the corresponding ToU costs as follows:

$$\begin{array}{ll} \Delta_1 = \Delta_3 = B, & c_1 = c_3 = 0; \\ \Delta_2 = 1, & c_2 = 1. \end{array}$$

Then it is clear that the answer to the question for instance $\mathcal{I}_1$ is "yes" if and only if the optimal solution value to the problem instance $\mathcal{I}_2$ is zero, which implies that no constant approximation exists. □

3. Algorithms for the basic model of simpler costs

Although our basic STOUC problem is very difficult in general as discussed in Section 2, some prominent special cases, of interest on their own, can be dealt with efficiently. In this section, we provide an optimal polynomial-time algorithm and a pseudo-polynomial-time algorithm for the basic STOUC problem when the cost vector has one and two valleys, respectively. We deal with another special case in the next section.

We say a schedule (resp. partial schedule) is *dense* (resp. a *dense block*) if there is no idle time between processing any two adjacent jobs. Clearly, any schedule consists of one or more dense blocks.

Lemma 1. *There exists an optimal schedule in which each dense block starts at a_i or ends at a_j for some $0 \le i, j \le m$.*

We provide our proofs of all lemmas in the Appendix for the continuity of our exposition.

When a dense block is processed from time t_1 to t_2, we say that the dense block occupies time interval $[t_1, t_2)$. If $[t_1, t_2) \cap [a_{i-1}, a_i) \neq \emptyset$ for some $1 \le i \le m$, we say that the block *uses* period $\mathcal{P}_i = [a_{i-1}, a_i)$.

Lemma 2. *If the cost vector π contains exactly $v \ge 1$ valley(s), then there exists an optimal schedule that consists of no more than v dense blocks and each dense block uses at least one valley.*

Using Lemmas 1 and 2, we are able to solve the basic STOUC problem efficiently when the cost vector π contains one valley.

Theorem 3. *The basic STOUC problem is solvable in polynomial time if cost vector π contains no more than one valley.*

Proof. According to Lemmas 1 and 2, there exists an optimal dense schedule that starts at a_i or ends at a_j, for some $0 \le i, j \le m$. To find the optimal schedule, we simply check all a_i for $0 \le i \le m$ as candidate start and end points of the schedule. The time complexity of the search is therefore $O(m)$. □

With Theorems 2 and 3, we have drawn a clear boundary for the complexity of our basic STOUC problem. Let us make the boundary even finer.

Theorem 4. *The basic STOUC problem is solvable in pseudo-polynomial time if π has two valleys.*

Proof. According to Lemmas 1 and 2, there are at most two dense blocks in an optimal schedule, and either dense block starts at some a_i or ends at some a_j, for some $0 \le i \le m-1$ and $1 \le j \le m$. If we know the lengths of these two dense blocks, we can find the optimal schedule by checking all a_i as candidate start and end points for these two dense blocks. The time complexity of the checking process is $O(m^2)$. Now we generate all possible lengths of the two dense blocks as follows. Let $F(L_1, L_2, j) = 0$, if there exists a job set $\mathcal{J}(L_1) \subseteq \mathcal{J}_j = \{J_1, \ldots, J_j\}$ such that $\sum_{k \in \mathcal{J}(L_1)} p_k = L_1$ and $\sum_{k \in \mathcal{J}_j / \mathcal{J}(L_1)} p_k = L_2$, and $F(L_1, L_2, j) = 1$, otherwise. Initially, we set $F(0, 0, 0) = 0$ and $F(x, y, 0) = 1$ for any $x \neq 0$ or $y \neq 0$. We apply the following recursion:

$$F(L_1, L_2, j) = \min\left\{F(L_1 - p_j, L_2, j-1), F(L_1, L_2 - p_j, j-1)\right\}. \tag{2}$$

There are $O(nP^2)$ states in $F(\cdot, \cdot, \cdot)$. Hence, the complexity of the above dynamic programming is $O(nP^2)$.

Since we have $O(P^2)$ pairs of $\{L_1, L_2\}$ for which $F(L_1, L_2, n) = 0$, and we find the minimum total cost for a schedule corresponding to each pair by setting the start and end points at values of a_i for

$0 \le i \le m$, the time complexity of the whole algorithm is $O(nP^2 + m^2P^2)$. □

Remark 1. Note that, as a special case of two-valley π, it is *pyramidal*, where the ToU cost monotonically increases to some point and then monotonically decreases, as considered by Fang et al. (2016). This special case can be solved more easily, by a pseudo-polynomial-time dynamic programming, since optimal dense blocks in this case must start at time 0 or end at time T.

Remark 2. According to Theorems 2 and 4, the basic STOUC problem with a two-valley ToU cost vector is neither in APX nor strongly NP-hard.

By extending the number of valleys in π or restricting the number of distinct job lengths in the above theorem, we immediately obtain the following conclusions.

Corollary 5. *The basic STOUC problem is solvable in pseudo-polynomial time if π has constant v valleys, and the time complexity is $O(nP^v + m^vP^v)$.* □

Corollary 6. *If π has two valleys and the number of distinct job sizes is r, then there is a polynomial time algorithm that optimally solves the basic STOUC problem and the time complexity is $O(m^2n^{2r-1})$.*

Proof. If the number of distinct job sizes is r, the number of distinct lengths of L_1 or L_2 in recursion (2) is bounded by $\binom{n+r-1}{r-1}$, which is polynomial in n. There are $O(n^{2r-1})$ states in $F(L_1, L_2, j)$. Using the algorithm in the proof of Theorem 4, we obtain $O(m^2n^{2r-1})$ as the time complexity of the algorithm. □

4. FPTAS for the basic model of bounded cost ratio

Let us further refine our picture for the solvability of the basic STOUC problem. In this section, we provide a fully polynomial-time approximation scheme (FPTAS) for the problem when the cost vector π has two valleys and the ToU cost ratio $\max_i c_i / \min_i c_i$ is bounded. Such a FPTAS basically involves partitioning of all the jobs into two blocks of suitable lengths each, approximating optimal solutions to two Subset-Sum problems.

4.1. Approximation of optimal block lengths

Denote by $F^*(\lambda_1, \lambda_2), F^*(\lambda) \in (0, +\infty]$ the minimum costs of processing two dense blocks of lengths λ_1 and λ_2, and one dense block of length λ, respectively. Such minimum costs can be determined in $O(m^2)$ and $O(m)$ time according to Lemma 1. Let σ^* be any optimal schedule with two dense blocks of lengths L_1^* and $P - L_1^*$, and let OPT be the cost of schedule σ^*. Then we have

$$\text{OPT} = F^*(L_1^*, P - L_1^*). \tag{3}$$

Now let us proximate the two optimal block lengths: L_1^* and $P - L_1^*$. Recall that the periods in $\mathcal{P}$ are indexed in the direction of time. Let their corresponding costs satisfy $c_{i_1} \le \cdots \le c_{i_m}$ for some index permutation $(i_1, \ldots, i_m)$ and let $\mathcal{P}_h$ be the hill between the two valleys of cost vector π for some $1 \le h \le m$. Denote $\kappa_v = \sum_{u=1}^{v} \Delta_{i_u} c_{i_u}$ for $v = 1, \ldots, m$. Determine $\underline{u}$ such that

$$\sum_{u=1}^{\underline{u}-1} \Delta_{i_u} < P \le \sum_{u=1}^{\underline{u}} \Delta_{i_u}, \quad 1 \le \underline{u} \le m. \tag{4}$$

We can assume without loss of generality that (a) the optimal schedule contains two dense blocks; (b) $\underline{u} \ge 2$ and $\text{OPT} \le \kappa_{\bar{u}}$ with $i_{\bar{u}} = h$, which is the index of the hill, since violation of either inequality implies that there is an optimal schedule of one dense block, which can be find in $O(m^2)$ time according to Lemma 1. Since it is evident that $\text{OPT} > \kappa_{\underline{u}-1}$, we focus on the following scenario:

$$\kappa_{\ell-1} < \text{OPT} \le \kappa_\ell \quad \text{for some fixed } \ell\text{: } \underline{u} \le \ell \le \bar{u}. \tag{5}$$

Recall that period $\mathcal{P}_h = [a_{h-1}, a_h)$. Let L_1 and L_2 be, respectively, the total *length* of periods in $\{\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_\ell}\}$ that fall into valley 1 (before a_{h-1}) and valley 2 (no earlier than a_{h-1}), namely,

$$L_1 = \sum_{u \le \ell:\, i_u < h} \Delta_{i_u}, \text{ and } L_2 = \sum_{u \le \ell:\, i_u \ge h} \Delta_{i_u}.$$

Note that: (a) the periods counting towards L_1 and L_2 are connected in their own valleys with period $\mathcal{P}_h$ in between; (b) if $\ell = \bar{u}$, then the length Δ_h of the hill period is counted towards L_2; (c) if at least one of L_1 and L_2 is zero, then it is evident that one dense block in the corresponding single valley is an optimal solution. Therefore, we assume $L_1, L_2 > 0$.

According to Lemma 2, either dense block in the optimal schedule uses a valley. Without loss of generality due to symmetry, we assume that dense block of length L_i^* uses the valley in L_i, $i = 1, 2$. The following two Subset-Sum problems will be useful:

$$L = \max \sum_{j=1}^{n} p_j x_j, \text{ s.t. } \sum_{j=1}^{n} p_j x_j \le L_1 \text{ and } x_j \in \{0, 1\}. \tag{6}$$

$$R = \max \sum_{j=1}^{n} p_j x_j, \text{ s.t. } \sum_{j=1}^{n} p_j x_j \le P - L_1 \text{ and } x_j \in \{0, 1\}. \tag{7}$$

Let $x^* = (x_1^*, \ldots, x_n^*)$ and $\hat{x}^* = (\hat{x}_1^*, \ldots, \hat{x}_n^*)$ be optimal solutions to (6) and (7), respectively. Given our definition of L_1, we can determine the value L_1^* in terms of L and R with the following lemma.

Lemma 3. *Suppose there is an optimal schedule of two dense blocks with the first block of length L_1^*. Then $L_1^* = L$ if $L_1^* \le L_1$ and $L_1^* = P - R$ otherwise.*

4.2. The details of the scheme

Note that (6) and (7) are two instances of the Subset-Sum problem, for which there is an FPTAS (see, e.g., Kellerer, Mansini, Pferschy, & Speranza, 2003). We establish that such an FPTAS, let us call it SS-FPTAS-Any, can be used here to otbain an FPTAS to our problem under the assumptions of (5) and that c_{i_m}/c_{i_1} is bounded by a constant. Let $\epsilon > 0$ be any given approximation precision required.

Subroutine SS(ℓ)

1. Use SS-FPTAS-Any to get a $(1-\epsilon)$-approximate solution $x' = (x'_j)$ to problem (6) and a $(1-\epsilon)$-approximate solution $\hat{x}' = (\hat{x}'_j)$ to problem (7).
2. Denote $\mathcal{J}' = \{J_j \in \mathcal{J} : x'_j = 1\}$ and $\hat{\mathcal{J}}' = \{J_j \in \mathcal{J} : \hat{x}'_j = 0\}$. Let schedule σ_1 (resp. σ_2) be such that all jobs in $\mathcal{J}'$ (resp. $\hat{\mathcal{J}}'$) are in one dense block positioned optimally in valley 1 and the remaining jobs are in one dense block positioned optimally in valley 2.
3. Output schedule $\sigma(\ell) \in \{\sigma_1, \sigma_2\}$ of the smaller cost.

Full scheme SS-FPTAS

1. Calculate k according to (4).
2. Compute $F^*(P)$ and the corresponding schedule σ_0. If $\underline{u} = 1$ or $i_{\underline{u}} > h$, then stop and output $\sigma := \sigma_0$.
3. For $\ell = \underline{u}, \ldots, \bar{u}$, call subroutine SS($\ell$) to produce schedule $\sigma(\ell)$. Output the schedules from $\{\sigma_0, \sigma(\underline{u}), \ldots, \sigma(\bar{u})\}$ that has the least execution cost.

Let us now establish that the scheme SS-FPTAS presented above is indeed an FPTAS.

Theorem 7. *The scheme SS-FPTAS is a fully polynomial-time approximation scheme for the basic STOUC problem when the cost vector has at most two valleys and the ratios of ToU costs are bounded by a constant.*

Proof. Denote by APP the total cost of executing the schedule produced by SS-FPTAS. We prove that

$$\frac{\text{APP}-\text{OPT}}{\text{OPT}} \le \left(\frac{c_{i_m}}{c_{i_1}}-1\right)\epsilon. \tag{8}$$

From the description of our scheme, we only need to prove the above inequality when (5) is satisfied. Recall that the optimal schedule σ^* consists of two dense blocks of lengths L_1^* and $P-L_1^*$, respectively. We need to consider two cases: $L_1^* \le L_1$ and $L_1^* > L_1$.

Case 1: $L_1^* \le L_1$. According to Lemma 3, $L_1^* = L$ in (6). Let $L_1' = \sum\{p_j, J_j \in \mathcal{J}'\}$, then it is $(1-\epsilon)$-approximate to L_1^*. We have $\delta \equiv L_1^* - L_1' \le \epsilon L_1^*$. According to SS($\ell$), we have APP $\le F^*(L_1', P-L_1')$. Therefore, using (3) we obtain

$$\begin{aligned}\text{APP}-\text{OPT} &\le F^*(L_1', P-L_1') - F^*(L_1^*, P-L_1^*)\\ &\le \delta(c_{i_m}-c_{i_1}) \le \epsilon L_1^*(c_{i_m}-c_{i_1}),\end{aligned}$$

where the second inequality is due to the fact that $(c_{i_m}-c_{i_1})$ is the maximum unit ToU cost difference. The above implies (8) due to the fact that $\text{OPT} \ge Pc_{i_1} \ge L_1^* c_{i_1}$.

Case 2: $L_1^* > L_1$. According to Lemma 3, $L_1^* = P-R$ in (7). Let $\hat{L}_1' = \sum\{p_j, J_j \in \hat{\mathcal{J}}'\}$ and $\hat{\delta} \equiv \hat{L}_1' - L_1^*$. Since $P-\hat{L}_1' \le R$ according to the definitions of $\hat{L}_1'$ and R, we have $\hat{\delta} \ge 0$. Since $P-\hat{L}_1'$ is $(1-\epsilon)$-approximate to R, i.e., $P-\hat{L}_1' \ge (1-\epsilon)R$, by rearranging the terms we obtain $\hat{\delta} \le \epsilon R$. According to SS($\ell$), we obtain APP $\le F^*(\hat{L}_1', P-\hat{L}_1')$. Therefore, using (3) we have

$$\begin{aligned}\text{APP}-\text{OPT} &\le F^*(\hat{L}_1', P-\hat{L}_1') - F^*(L_1^*, P-L_1^*)\\ &\le \hat{\delta}(c_{i_m}-c_{i_1}) \le \epsilon R(c_{i_m}-c_{i_1}),\end{aligned}$$

which implies (8) due to the fact that $\text{OPT} \ge Pc_{i_1} \ge Rc_{i_1}$. □

5. Extensions of the basic model

Having studied thoroughly the basic STOUC problem, let us move on to some natural extensions of the basic model. While our STOUC problem is still to seek a *feasible* schedule of minimum total execution cost, the feasibility requirement will be more general or more complicated than a simple deadline. In this section, we investigate the following feasibility requirements: bounded (a) lateness (or tardiness), (b) flow-time, and (c) total of job completion times.

Since the deadline condition did not play any explicit role in the proofs of Theorems 1 and 2, the hardness of the STOUC problem with any of the above three feasibility requirements remains, as can be seen easily by specifying the feasibility bound large enough to make the feasibility requirement non-binding.

5.1. Bounded lateness

The *lateness* of job J_j is defined as $\ell_j = C_j - d_j$, where d_j is, as introduced at the beginning of Section 2, the due time of the job. Similar to the basic STOUC problem, we require that the lateness of any job is bounded above by a pre-specified value, which is part of the problem input. We start with an identification, in the following lemma, of optimal schedules of some special structure.

Lemma 4. *When the cost vector π has one valley, there is an optimal schedule in which jobs are scheduled in order of non-decreasing due times (EDD for short of earliest-due-date) and jobs of the same due times can be in any order.*

Remark 3. Note that NP-hardness of a scheduling problem does not necessarily imply nonexistence of an optimal schedule of EDD order. Consequently, it is worth noticing that the condition of one-valley cost vector is needed in Lemma 4. Otherwise, EDD order can be suboptimal as the following example shows: There are $m=3$ periods with $\Delta_1 = 2, \Delta_2 = \Delta_3 = 1$ and $c_1 = c_3 = 0, c_2 = 1$. There are two jobs J_1 and J_2 with their processing times and due times $p_1 = d_1 = 1$ and $p_2 = d_2 = 2$. Feasibility requires that lateness be up bounded by 3. Then the optimal schedule is to schedule job J_2 in period $\mathcal{P}_1$ and J_1 in period $\mathcal{P}_3$ with a total execution cost of 0, while any schedule of EDD order has a cost of 1.

According to Lemma 4, we assume for the rest of this subsection that jobs are indexed in EDD order, since there is an optimal schedule that is in this EDD order. Define

$$Q_{kst}^- = a_k - P_{st},\ Q_{kst}^+ = a_k + P_{st};\quad 0 \le k \le m,\ 1 \le s,t \le n; \tag{9}$$

and

$$\bar{Q}_{jst}^- = (d_j + \bar{L}_{\max}) - P_{st},\ \bar{Q}_{jst}^+ = (d_j + \bar{L}_{\max}) + P_{st};\quad 1 \le j,s,t \le n; \tag{10}$$

where the P_{st} values are defined in (1) and $\bar{L}_{\max}$ is the pre-specified upper bound on job lateness as part of the problem input. Merge the two sets of *non-negative* Q values in (9) and the two sets of *non-negative* $\bar{Q}$ values in (10), arrange them in ascending order and then denote all these values as $\lambda_0 = a_0 < \lambda_1 < \cdots < \lambda_N = a_m$. It is easy to see that $N = O((m+n)n^2)$. According to the definition, time interval $[\lambda_{i-1}, \lambda_i)$ is contained in a unique period of $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$. Similar to Lemma 1, the following lemma describes some specific structure of an optimal schedule. We provide a brief proof of the lemma in the appendix, which is based on the idea of moving jobs to certain starting times, very similar to that in the proof of Lemma 1.

Lemma 5. *When cost vector π has one valley, there is an optimal schedule (with bounded lateness) in which jobs are sequenced $(1, \ldots, n)$ and the completion time of any job is equal to λ_t for some $1 \le t \le N$.*

Using Lemma 5, we can solve the corresponding problem efficiently by dynamic programming.

Theorem 8. *The STOUC problem with bounded lateness is solvable in $O(m(m+n)n^3)$ time, when the cost vector has one valley.*

Proof. Index the jobs according to EDD order. The feasibility of the problem can be easily checked by scheduling all jobs in one dense block and starting it at time 0. Let $f(j, t)$ be the minimum total cost for processing jobs $J_1, \ldots, J_j$ subject to the condition that J_j is completed by time t. According to Lemma 5, we can focus on $t \in \{\lambda_1, \ldots, \lambda_N\}$. Let $\lambda_j' = a_0 + P_{1j}$ and $\lambda_j'' = \min\{a_m - P_{j+1,n}, d_j + \bar{L}_{\max}\}$. Clearly, $\lambda_j', \lambda_j'' \in \{\lambda_1, \ldots, \lambda_N\}$ are respectively the minimum and maximum values of t for processing jobs $J_1, \ldots, J_j$ when J_j is completed by time t in any feasible schedule. With the definitions of λ_j' and λ_j'', for all $\lambda_i \notin (\lambda_j', \lambda_j'']$ $(j = 1, \ldots, n)$ we have:

$$f(j, \lambda_i) = \begin{cases} \infty, & \text{for } \lambda_i < \lambda_j',\\ \sum_{t=1}^{\lambda_i} c(t), & \text{for } \lambda_i = \lambda_j',\\ f(j, \lambda_j''), & \text{for } \lambda_i > \lambda_j''. \end{cases} \tag{11}$$

Note in particular that $f(j, \lambda_j')$ for all $j = 1, \ldots, n$ are determined without the need of recursive computation. The initialization of our dynamic programming algorithm is therefore completed for all

$\lambda_i \in (\lambda'_1, \lambda''_1]$ as follows:

$$f(1,\lambda_i) = \min_{s:\, s\le i \wedge \lambda_s \ge \lambda'_1} \left\{ \sum_{t=1}^{p_1} c(\lambda_s - p_1 + t) \right\}, \text{ for } \lambda'_1 < \lambda_i \le \lambda''_1. \tag{12}$$

Recall that $c(\cdot)$ in the above formula is a ToU cost defined at the beginning of Section 2. Noticing that we have (11) for all $\lambda_i \notin (\lambda'_j, \lambda''_j]$, we use the following forward recursion for any $j = 2, \ldots, n$:

$$f(j,\lambda_i) = \min \left\{ \sum_{t=1}^{p_j} c(\lambda_i - p_j + t) + f(j-1, \lambda_i - p_j), f(j, \lambda_{i-1}) \right\},$$
$$\text{for } \lambda'_j < \lambda_i \le \lambda''_j. \tag{13}$$

On the running time of our DP algorithm (11)–(13), there are $O(nN)$ possible values for $f(j, \lambda_i)$. To evaluate each of (12) and (13), we need to calculate the value $\sum_{t=t_1}^{t_2} c(t)$ for some pair of t_1 and t_2 such that $t_1 \le t_2$ with $t_1, t_2 \in [a_0, a_m]$, which can be done in $O(m)$ time by first determining a pair of indices $u \le v$ such that $t_1 \in \mathcal{P}_u$ and $t_2 \in \mathcal{P}_v$. Therefore, we need a total of $O(nmN) = O(m(m+n)n^3)$ time for implementation of the dynamic programming. □

Remark 4. If all periods $\mathcal{P}_i$ have a unit length, $\Delta_i = 1$ for all $i = 1, \ldots, m$, then the time complexity of the dynamic programming is $O(nm^2)$.

5.2. Bounded tardiness, delivery- or flow-time

The *tardiness* of job J_j is defined as the lateness ℓ_j of the job unless $\ell_j < 0$, in which case it is defined as zero. Note that any schedule minimizing maximum lateness also minimizes maximum tardiness (but not *vice versa*). Consequently, the STOUC problem with bounded tardiness can be reduced to the STOUC problem with bounded lateness (with the same bound), which we have studied in Section 5.1. However, the converse is not true, when the bound on the lateness is a negative number $-\alpha$, which indicates the requirement that each job must be completed at least α time units in advance of its due time.

Now suppose each job J_j needs to be delivered, which takes $q_j \ge 0$ time units, after it is completed at C_j on the machine. A schedule is feasible if all jobs are delivered to the customers by a common deadline $D \ge q_j$ $(j = 1, \ldots, n)$. We show that this problem, called STOUC-D, is equivalent to the STOUC problem with bounded tardiness, called STOUC-T. Given any instance of STOUC-D with bound D on the delivery time $C_j + q_j$ of each job J_j, we generate an instance of STOUC-T by simply setting the due time of job J_j as $d_j = D - q_j \ge 0$ and setting the tardiness bound as $T = 0$. Clearly, a schedule is feasible for STOUC-T if and only if this schedule is feasible for STOUC-D. On the other hand, given any instance of STOUC-T with bound $T \ge 0$ on the tardiness $\max\{0, C_j - d_j\}$ of each job J_j, we generate an instance of STOUC-D by simply setting the delivery time of job J_j as $q_j = d_{\max} - d_j \ge 0$, where $d_{\max} = \max_j d_j$, and setting the delivery bound $D = T + d_{\max} \ge q_j$. Clearly, a schedule is feasible for STOUC-D if and only if this schedule is feasible for STOUC-T.

The *flow-time* of job J_j is defined as $C_j - r_j$, where r_j is, as introduced at the beginning of Section 2, the release time of the job. For the STOUC problem with bounded flow-time, results parallel to Lemmas 4 and 5 and Theorem 8, which are on bounded lateness, can be established in a straightforward way by replacing due time d_j with release time r_j in the corresponding analyses.

5.3. Bounded total of job completion times

Having studied the STOUC problem with three bounded-max feasibility requirements, let us consider the problem with bounded-sum feasibility requirement: in any feasible schedule, the total of job completion times should not exceed a pre-specified upper bound.

Given the clear picture of computational complexity already established in Section 2 for the problem when the ToU cost vector has more than one valley, we focus the situation of one valley. As Lemma 4, the following lemma identifies an optimal job sequence.

Lemma 6. *When the cost vector π has one valley, there is an optimal schedule in which jobs are scheduled in order of non-decreasing processing times (SPT for short of shortest-processing-time first).*

Remark 5. As in Remark 3, we note that NP-hardness of a scheduling problem does not necessarily imply nonexistence of an optimal schedule of SPT order. Consequently, it is worth noticing that the condition of one-valley cost vector is needed in Lemma 6. Otherwise, SPT order can be suboptimal as the following example shows: There are $m = 3$ periods with $\Delta_1 = 2, \Delta_2 = \Delta_3 = 1$ and $c_1 = c_3 = 0, c_2 = 1$. There are two jobs with $p_1 = 1$ and $p_2 = 2$. Let the total of job completion times be up bounded by 6. Then the optimal schedule is to schedule job J_2 in period $\mathcal{P}_1$ and J_1 in period $\mathcal{P}_3$ with a total cost of 0, while any schedule of SPT order has a cost of 1.

Using Lemma 6, we assume in this subsection that jobs are indexed in SPT order, with which and (9) we arrange all *non-negative* Q values in ascending order and denote them as $a_0 = b_0 < b_1 < \cdots < b_{\tilde{N}} = a_m$. We will call these numbers b-values. Then it is easy to see that $\tilde{N} = O(mn^2)$. According to the definition, time interval $[b_{i-1}, b_i)$ is contained in a unique period of $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$. We say a dense block *non-floating* if each of its job start (completion) times is equal to some b_t and floating otherwise. It is easy to see that for any dense block, if one of the jobs is non-floating, then all are non-floating. The following lemma describes some specific structure of an optimal schedule.

Lemma 7. *When cost vector π has one valley, there is an optimal schedule with bounded total of job completion times in which (a) jobs are sequenced $(1, \ldots, n)$, (b) there are at most m dense blocks, and (c) all dense blocks are non-floating except possibly one. If such a floating block exits, then the bound on the total of job completion times is binding.*

Having the nice structures of an optimal schedule described in Lemma 7, we are in a position to solve the STOUC problem at hand with some efficiency.

Theorem 9. *When the ToU cost vector has one valley, the STOUC problem with bounded total of job completion times is solvable in $O(m^{m+2}n^{3m-1})$ time, which is a polynomial when m is fixed.*

Proof. Index the jobs in any SPT order. According to Lemma 7, there is an optimal schedule of job sequence $(1, \ldots, n)$ that has *exactly* m dense blocks with some of the neighboring pairs possibly merged together. Clearly, there are $\binom{n-1}{m-1}$ ways to partition the n jobs into m dense blocks while keeping the SPT order. Given any such partition, there is at most one floating dense block in an optimal schedule according to Lemma 7. Let B_i be the length of the ith dense block in this partition, $i = 1, \ldots, m$, and assume that the kth dense block is potentially floating for some $1 \le k \le m$. For the other $m-1$ non-floating dense blocks, we assign $m-1$ start times $\{t_i : 1 \le i \le m, i \ne k\} \subseteq \{b_0, \ldots, b_{\tilde{N}-1}\}$. There are $\binom{\tilde{N}}{m-1}$ ways to do the assignment since these start times must be ordered increasingly. If we denote $t_{m+1} = T$, then the feasibility conditions for the

start time assignment is as follows:

$$\begin{cases} t_{j+1} - t_j \geq B_j, \text{ for } j \in \{1, \ldots, m\} \setminus \{k-1, k\}; \\ t_{k+1} - t_{k-1} \geq B_{k-1} + B_k; \\ \bar{t}_k \equiv \bar{\mu} - \sum_{j \neq k}(t_j + B_j) \geq t_{k-1} + B_{k-1} + B_k; \end{cases}$$

where $\bar{\mu}$ is the pre-specified upper bound on the total of job completion times. Then, the start time of the kth block can be assigned to one of the b-values of the interval $[t_{k-1} + B_{k-1}, t_{k+1} - B_k]$ or to $\bar{t}_k$ if $\bar{t}_k \leq t_{k+1} - B_k$, in which case the upper bound of $\bar{\mu}$ is binding. The total number of start time assignments of all possible m dense blocks is then bounded by $\binom{n-1}{m-1} \times m \times \binom{\tilde{N}}{m-1} \times \tilde{N}$, which is $O(m^{m+1} n^{3m-1})$.

Once the start times of all three dense blocks are assigned and the feasibility conditions displayed above are checked, we can calculate in $O(m)$ time the execution cost of the schedule corresponding to the assignment. Therefore, the total time to find an optimal schedule is $O(m^{m+2} n^{3m-1})$. □

Remark 6. It is not difficult to see that the assumption of one valley in the ToU costs in Lemma 7 and hence in Theorem 9 can be relaxed to any constant number of valleys, for bounding by a constant the number of floating dense blocks in an optimal schedule. The one-valley assumption is necessary, however, for the existence of an optimal schedule in which jobs are sequenced in SPT order, as we have demonstrated in Remark 5. Also the bound m on the number of dense blocks is tight in the above theorem.

6. Concluding remarks

As demonstrated in Section 2, our STOUC problem is computationally intractable even for very restricted ToU costs, which form only two valleys. We have also shown in other parts of the paper various ways of restricting the structure of the ToU costs to make the STOUC problem tractable.

We can further identify some special cases of the STOUC problem for which efficient algorithms exist or can be found. For example, if the ToU costs are non-decreasing, then the STOUC problem with bounded number of late jobs is solvable in $O(n \log n)$ time by the Moore–Hodgson algorithm (Moore, 1968), where a job J_j is late if its lateness $\ell_j = C_j - d_j$ is positive.

Nonetheless, with this paper we have achieved our primary objective of drawing a fine boundary of solvability of the STOUC problem. We hope our study will stimulate further research on scheduling problems with ToU costs.

Acknowledgments

This work is supported to the first author in part by a Visiting Professorship of the School of Management, Fudan University; and to the second author in part by the National Natural Science Foundation of China (Grant No. 11371103, 71531005 and 71222104) and a Shuang-Yi-Liu Grant to Department of Management Science, Fudan University.

Appendix

A.1. Proof of Lemma 1

Assume first that there is only one dense block in the optimal schedule, which starts at t_0 and ends at t_1, where $a_{i-1} < t_0 < a_i$ and $a_{j-1} < t_1 < a_j$ for some $1 \leq i, j \leq m$.

Case 1: $c_i \leq c_j$. If $t_0 - a_{i-1} \leq t_1 - a_{j-1}$, we reschedule the dense block to start at a_{i-1}. Otherwise, we reschedule the dense block to end at a_{j-1}. In either situation, the total execution cost is not increased.

Case 2: $c_i > c_j$. If $a_i - t_0 \leq a_j - t_1$, we reschedule the dense block to start at a_i. otherwise, we reschedule the dense block to end at a_j. In either situation, the total execution cost is not increased.

If the optimal schedule consists of more than one dense block, the above argument still applies with only some minor adjustment to take account occupation of neighboring dense block(s).

A.2. Proof of Lemma 2

Let us prove the lemma by induction. Consider $\nu = 1$. Let period $\mathcal{P}_k = [a_{k-1}, a_k)$ be the valley. Given any optimal schedule, if it is not dense, let us consider the first two (adjacent) dense blocks, which occupy time intervals $[t_1, t_2)$ and $[t_3, t_4)$, respectively, with $t_2 < t_3$.

If $t_3 < a_k$, we reschedule the first dense block to $[t_3 - (t_2 - t_1), t_3)$. This will not increase the total execution cost, because the ToU cost is non-increasing before a_k. As a result, we have an optimal schedule with one dense block less. Symmetrically, we can do this if $t_2 \geq a_{k-1}$, in which case we reschedule the second dense block to $[t_2, t_2 + (t_4 - t_3))$. Finally, if $t_3 \geq a_k$ and $t_2 < a_{k-1}$, we can reduce the number of dense blocks in the optimal schedule by rescheduling its first dense block to $[a_{k-1} - (t_2 - t_1), a_{k-1})$ and its second dense block to $[a_k, a_k + (t_4 - t_3))$. Therefore, after at most $n-1$ repetitions of the above merger process, we obtain an optimal schedule that is dense.

Let the (resulting) dense block occupy interval $[t_a, t_b)$. If it does not use $\mathcal{P}_k$, we have $t_a > a_k$ or $t_b < a_{k-1}$. In the former case, we reschedule the dense block to $[a_{k-1}, a_{k-1} + (t_b - t_a))$. In the latter case, we reschedule the dense block to $[a_k - (t_b - t_a), a_k)$. Apparently, such a rescheduling will not increase the total execution cost.

Now let us consider $\nu \geq 2$. If an optimal schedule σ consists of more than ν dense blocks, we construct an optimal schedule of at most ν dense blocks in the following way. Let $\mathcal{P}_h = [a_{h-1}, a_h)$ be the hill between the first two valleys. Consider the time intervals $\mathcal{T}_1 = [0, a_{h-1})$ and $\mathcal{T}_2 = [a_{h-1}, T)$. Within time intervals $\mathcal{T}_1$ and $\mathcal{T}_2$, there are exactly one valley and $\nu - 1$ valleys, respectively. According to our induction hypothesis, jobs (if any) scheduled under σ to be processed, respectively, *within* $\mathcal{T}_1$ and $\mathcal{T}_2$ can be rescheduled (without increasing the total execution costs) to form at most one and at most $\nu - 1$ dense blocks, respectively, and each block uses at least one valley. If there is a job J_j scheduled (under σ) to occupy time interval $[t_1, t_2)$ with $a_{h-1} \in (t_1, t_2)$, let $t_1 \in \mathcal{P}_a$ and $t_2 \in \mathcal{P}_b$ for some $a, b \in \{1, \ldots, m\}$. Then we reschedule job J_j to the end of the first block when $c_a \leq c_b$ and to the beginning of the second block otherwise. Consequently, we obtain an optimal schedule with at most k dense blocks and each dense block uses at least one valley.

A.3. Proof of Lemma 3

Let $\bar{c}$ be the highest ToU cost among those corresponding to the periods in the first valley whose lengths sum to L_1. Then $\bar{c} < c_\ell$ according to our construction of L_1 and L_2.

If $L_1^* \leq L_1$, then we must have $\delta \equiv L - L_1^* = 0$. Otherwise, $\delta > 0$, we could obtain a schedule of smaller total execution cost than σ^* as follows: schedule all jobs in $\{J_j : x_j^* = 1\}$ as a dense block optimally in the first valley and schedule the remaining jobs as a dense block optimally in the second valley. The total execution cost of the resulting schedule differs from the value in (3) by at most $\delta \bar{c} - \delta c_\ell < 0$.

On the other hand, suppose $L_1^* > L_1$. Denote $\hat{L} = P - R$. Then $\hat{\delta} \equiv L_1^* - \hat{L} \geq 0$ by definition. If $\hat{\delta} > 0$, then we can obtain another optimal schedule σ' whose first dense block has a length equal to $\hat{L}$: schedule all jobs in $\{J_j : \hat{x}_j^* = 1\}$ as a dense block optimally in the

first valley and schedule the remaining jobs as a dense block in the second valley. We see that, comparing with σ^*, the new schedule σ' decreases (resp. increases) the length of the first (resp. second) dense block by $\hat{\delta}$. Note that if the first dense block of schedule σ^* uses a period $\mathcal{P}_i$ that contributes to the length of L_2, then the block does so by first using the hill $\mathcal{P}_h$ and the ToU cost c_i is at least the ToU cost of any period used by the dense block $P - L_1^*$. Therefore, changing from σ^* to σ', the decreased cost due to the decreased length of the first dense block is at least as large as the increased cost due to the increase of the second dense block.

A.4. Proof of Lemma 4

The proof of the lemma is based on a simple exchange argument. Given any optimal schedule σ, it is evident that we can assume without loss of generality that there is no idle time from the start time τ of the valley period. For any neighboring pair of jobs (in schedule σ) not in EDD order or of the same due time, let us change σ to another optimal schedule σ' by simply changing the order of these two jobs while keeping all the other jobs intact. More specifically, let (J_j, J_k) be any pair of neighboring jobs in this order with $d_j \geq d_k$. If there is no idle time between the two jobs, then we are done—the larger one of the two job latenesses decreases in the resulting schedule. So assume there is some idle time between the two jobs, which implies that job J_k starts no later than time τ. Let the two jobs have completion times of $C_j(\sigma)$ and $C_k(\sigma)$, respectively, in schedule σ. While keeping all other jobs intact, we create a new schedule σ' by changing the order of the two jobs so that J_j finishes at $C_k(\sigma)$ (i.e., $C_j(\sigma') = C_k(\sigma)$) and J_k finishes at $C_k(\sigma') = C_k(\sigma) - p_j$. It is easy to see that the total execution cost of the resulting schedule σ' does not increase and hence must also be optimal.

A.5. Proof of Lemma 5

Given any schedule, if the completion time of a job is equal to λ_t for some $1 \leq t \leq N$, we say that the job is *non-floating*, otherwise it is *floating*. It is easy to see from the definition that for any dense block of a feasible schedule, if one of the jobs is non-floating, then all are non-floating. Therefore, we can say whether a whole block is floating or not. Let σ be an optimal schedule that has the minimum number of floating jobs. We show that σ has no floating jobs at all. Suppose to the contrary that σ has at least one floating job and let job J_j have the smallest index among all floating jobs in σ. Let $t_0 = C_{j-1}(\sigma)$ be the completion time of the preceding job J_{j-1} (if any) and $t_1 = S_j(\sigma)$ be the start time of job J_j. Then $t_0 < t_1$ since job J_j is floating. Let jobs $J_j, \ldots, J_k$ be the dense block of schedule σ for some $k \geq j$, which we call block $\mathcal{B}$, and denote $t_2 = C_k(\sigma)$ and $t_3 = S_{k+1}(\sigma)$ (which is equal to T if $k = n$), where $t_2 < t_3$.

Using the argument (very similar to what in the proof of Lemma 2 with $k = 1$) of moving two neighboring dense blocks towards each other without increasing the total execution cost and noticing that forward moving of a dense block keeps feasibility and backward moving can make the block non-floating before merging with the succeeding block, we conclude that either (a) block $\mathcal{B}$ merges with the preceding one if any, or (b) at least one of the jobs, and hence all jobs of block $\mathcal{B}$, become non-floating, or (c) block $\mathcal{B}$ merges with the succeeding one (if any) that is also floating. However, either (a) or (b) implies that all jobs in block $\mathcal{B}$ become non-floating in a new optimal schedule and hence it has less number of floating jobs, which is impossible due to the choice of schedule σ. This merger process continues until we get an optimal schedule in which jobs $J_j, \ldots, J_n$ form a dense block and all these jobs are floating. One more application of the argument of moving forward or backward leads to a final contradiction.

A.6. Proof of Lemma 6

As in the proof of Lemma 4, we use a simple exchange argument for the proof. Given any optimal schedule σ, we can assume without loss of generality that there is no idle time since the start time τ of the valley period. For any neighboring pair of jobs (in schedule σ) not in SPT order, let us change σ to another optimal schedule σ' by simply changing the order of these two jobs while keeping all the other jobs intact. More specifically, let jobs J_j and J_k be any pair of neighboring jobs with $j < k$ and $p_j > p_k$. If there is no idle time between the two jobs, then we are done—the total of the two job completion times decreased (by $p_j - p_k$). So assume there is some idle time between the two jobs, which in turn implies that job J_k starts no later than time τ. Let the two jobs have completion times of $C_j(\sigma)$ and $C_k(\sigma)$, respectively, in schedule σ. While keeping all other jobs intact, we create a new schedule σ' by changing the order of the two jobs so that J_j finishes at $C_k(\sigma)$ (i.e., $C_j(\sigma') = C_k(\sigma)$) and J_k finishes at $C_k(\sigma') = \min\{C_k(\sigma) - p_j, C_j(\sigma)\}$. It is easy to see that the total execution cost of the resulting schedule σ' does not increase and hence must also be optimal.

A.7. Proof of Lemma 7

As in the proof of Lemma 5, the proof here is mainly based on the simple idea of moving jobs to start at some special values, conclusion (a) follows from Lemma 6. Let σ be an optimal schedule with job sequence $(1, \ldots, n)$ such that it first has the minimum number ℓ_B of dense blocks and then has the minimum number ℓ_F of floating dense blocks. It is easy to see that $\ell_B \leq m$ since in each period $\mathcal{P}_i \in \mathcal{P}$, there can be at most one idle interval (between two dense blocks), otherwise any block between two such idle intervals can be moved forward to merge with the preceding dense block without changing the total execution cost, leading to a schedule of $\ell_B - 1$ dense blocks, which contradicts the choice of schedule σ.

Now let us focus on proving $\ell_F \leq 1$. Suppose to the contrary that σ has at least two floating dense blocks. Let blocks $\mathcal{B}, \mathcal{B}' \subseteq \mathcal{J}$ be the first (in terms of time) two floating blocks. Let b_s, b_t (resp. $b_{s'}, b_{t'}$) be the first and last b-values in block $\mathcal{B}$ (resp. $\mathcal{B}'$). (NB: these b-values must exist according to their definition and $s' \geq t + 1$.) Denote u (resp. v) and u' (resp. v') as the start (resp. end) times of the two blocks. Then $v < u'$ and none of $\{u, v, u', v'\}$ is a b-value. It is evident that the start time τ of the valley is at least $b_{s'}$ ($\tau \geq b_{s'}$, otherwise, block $\mathcal{B}'$ could move forward (i.e., to the left), either to become non-floating or to merge with the preceding block, resulting one less floating block). As illustrated in Fig. 2, we have

$$b_{s-1} < u < b_s \leq b_t < v < u' < b_{s'} \leq b_{t'} < v' < \tilde{b},$$

and $v < b_{t+1} \leq b_{s'-1} < u'$ if $s' \geq t + 2$, where $\tilde{b} = b_{t'+1}$ or $\tilde{b} < b_{t'+1}$ and it is the start time of the third floating dense block.

Now let us move the two floating blocks to create a new optimal schedule in which either the number of dense blocks or the number of floating blocks is strictly decreased, contradicting our choice of schedule σ. Let

$$\varepsilon = c(b_s) - c(b_{t+1}) \quad \text{and} \quad \varepsilon' = c(b_{s'}) - c(b_{t'+1}).$$

Then $\varepsilon \geq 0$ (due to the position of the valley) and $\varepsilon' \geq 0$ (otherwise, block $\mathcal{B}'$ could move forward either to become non-floating or to merge with the preceding block, resulting one less floating block). Let where $k = |\mathcal{B}|$ and $k' = |\mathcal{B}'|$ be the numbers of jobs in blocks $\mathcal{B}$ and $\mathcal{B}'$, respectively. Denote $\theta = k'/(k + k')$, $\eta = \theta r$ and

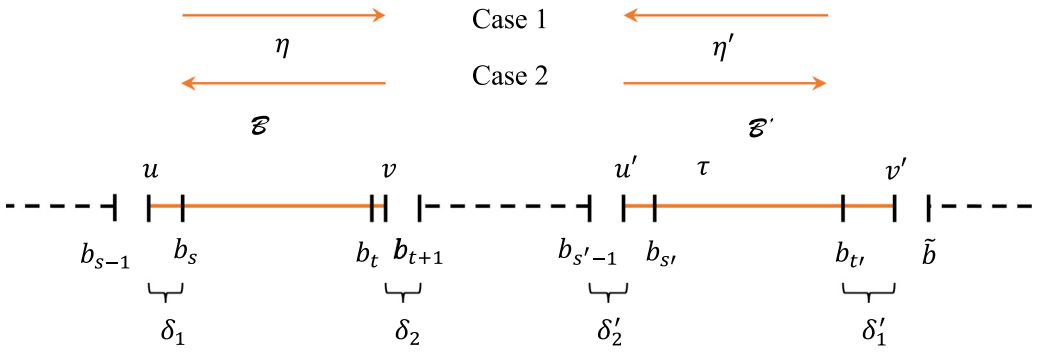

Fig. 2. Illustration of two floating blocks in schedule σ and their adjusting moves.

$\eta' = (1-\theta)r$, where

$$r = \begin{cases} \min\left\{\dfrac{\min\{\delta_1,\delta_2\}}{\theta}, \dfrac{\min\{\delta_2',\delta_1'\}}{1-\theta}\right\}, & \text{if } s' \geq t+2; \\ \min\left\{\dfrac{\min\{\delta_1, u'-v\}}{\theta}, \dfrac{\min\{u'-v,\delta_1'\}}{1-\theta}\right\}, & \text{if } s' = t+1; \end{cases}$$

and $\delta_1 = b_s - u$, $\delta_1' = v' - b_{t'}$; while $\delta_2 = b_{t+1} - v$, $\delta_2' = u' - b_{s'-1}$ if $s' \geq t+2$. Let us separate our consideration into two cases: either $\eta\varepsilon \geq \eta'\varepsilon'$ or otherwise.

Case 1: $\eta\varepsilon \geq \eta'\varepsilon'$.

There is at least one dense block between $[v, u')$ *or* $\eta + \eta' \leq u' - v$. In the case of $s' \geq t+2$, if $\min\{\delta_1,\delta_2\}/\theta \leq \min\{\delta_2',\delta_1'\}/(1-\theta)$, we have $\eta = \min\{\delta_1,\delta_2\}$ and $\eta' = \min\{\delta_1,\delta_2\}(1-\theta)/\theta \leq \min\{\delta_2',\delta_1'\}$. If $\min\{\delta_1,\delta_2\}/\theta > \min\{\delta_2',\delta_1'\}/(1-\theta)$, we have $\eta' = \min\{\delta_2',\delta_1'\}$ and $\eta = \min\{\delta_2',\delta_1'\}\theta/(1-\theta) < \min\{\delta_1,\delta_2\}$. In the case of $s' = t+1$, similar results can be found.

Therefore, on the one hand, we move block $\mathcal{B}$ backward (to the right) by η, which reduces the execution cost of the block by $\eta\varepsilon$ and also increases the total of job completion times by $k\eta$. On the other hand, we move block $\mathcal{B}'$ forward (to the left) by η', which increases the execution cost of the block by $\eta'\varepsilon'$ and also decreases the total of job completion times by $k'\eta'$. Consequently, the net change of execution cost of the resulting schedule is $\eta'\varepsilon' - \eta\varepsilon \leq 0$, while the net change of the total of job completion times is $k\eta - k'\eta' = 0$, which implies that the resulting schedule is also optimal. However, the new optimal schedule has at least one less floating block, which contradicts the choice of the original schedule σ.

The machine is idle between $[v, u')$ *and* $\eta + \eta' > u' - v$. Let $\xi = (u'-v)/(\eta+\eta')$. Then, move block $\mathcal{B}$ backward by $\xi\eta$ and move block $\mathcal{B}'$ forward by $\xi\eta'$. Using similar argument above, we get a new optimal schedule that has one less floating block.

Case 2: $\eta\varepsilon < \eta'\varepsilon'$.

In this case, we move the two floating blocks in the directions opposite, respectively, to the ones in Case 1 (as indicated in Fig. 2), although the values of η and η' are adjusted with the following new definitions of the δ-values:

$$\delta_1 = u - b_{s-1},\ \ \delta_2 = v - b_t,\ \ \delta_2' = b_{s'} - u',\ \ \delta_1' = \tilde{b} - v'.$$

Then with symmetric calculation, we obtain a new optimal schedule that has at least one less floating block, with which we have proved (c) in the lemma.

References

Allahverdi, A., & Aydilek, H. (2014). Total completion time with makespan constraint in no-wait flowshops with setup times. *European Journal of Operational Research, 238*(3), 724–734.

Badinelli, R. D. (2000). An optimal, dynamic policy for hotel yield management. *European Journal of Operational Research, 121*(3), 476–503.

Bitran, G., & Caldentey, R. (2003). An overview of pricing models for revenue management. *Manufacturing & Service Operations Management, 5*(3), 203–229.

Braithwait, S., Hansen, D., & O'Sheasy, M. (2007). Retail electricity pricing and rate design in evolving markets. *Technical report*. Edison Electric Institute.

Chawla, S., Devanur, N. R., Holroyd, A. E., Karlin, A. R., Martin, J., & Sivan, B. (2017). Stability of service under time-of-use pricing. In *Proceedings of the eighteenth ACM conference on economics and computation* https://www.microsoft.com/en-us/research/wp-content/uploads/2016/12/cloud.pdf. Cambridge, MA, USA; June 26–30.

Chen, B., Potts, C. N., & Woeginger, G. J. (1998). A review of machine scheduling: Complexity, algorithms and approximability. In D. Z. Du, & P. Pardalos (Eds.), *Handbook of combinatorial optimization: 3*. Kluwer Academic Publishers. 21–169.

Chen, C. L., & Bulfin, R. L. (1993). Complexity of single machine, multi-criteria scheduling problems. *European Journal of Operational Research, 70*(1), 115–125.

EIA (2018). Energy explained: Your guide to understanding energy. [online]. https://www.eia.gov/energyexplained/, accessed on November 15, 2018.

Fang, K., Uhan, N. A., Zhao, F., & Sutherland, J. W. (2016). Scheduling on a single machine under time-of-use electricity tariffs. *Annals of Operations Research, 238*, 199–227.

Geng, Z., & Yuan, J. J. (2015). Pareto optimization scheduling of family jobs on a p–batch machine to minimize makespan and maximum lateness. *Theoretical Computer Science, 570*, 22–29.

Gupta, D., & Denton, B. (2008). Appointment scheduling in health care: Challenges and opportunities. *IIE Transactions, 40*, 800–819.

Gupta, J. N. D., & Ruiz-Torres, A. J. (2000). Minimizing makespan subject to minimum total flow-time on identical parallel machines. *European Journal of Operational Research, 125*(2), 370–380.

Hoogeveen, H. (2005). Multicriteria scheduling. *European Journal of Operational Research, 167*(3), 592–623.

Hoogeveen, H., & de Velde, S. L. V. (1995). Minimizing total completion time and maximum cost simultaneously is solvable in polynomial time. *Operations Research Letters, 17*(5), 205–208.

Huo, Y., & Zhao, H. (2015). Total completion time minimization on multiple machines subject to machine availability and makespan constraints. *European Journal of Operational Research, 243*(2), 547–554.

Kasilingam, R. G. (1997). Air cargo revenue management: Characteristics and complexities. *European Journal of Operational Research, 96*(1), 36–44.

Kellerer, H., Mansini, R., Pferschy, U., & Speranza, M. G. (2003). An efficient fully polynomial approximation scheme for the subset-sum problem. *Journal of Computer and System Sciences, 66*(2), 349–370.

Kulkarni, J., & Munagala, K. (2013). Algorithms for cost-aware scheduling. *Lecture Notes in Computer Science, 7846*, 201–214.

Leung, J. Y.-T. (Ed.) (2004). Handbook of scheduling: Algorithms, models, and performance analysis. New York, USA: Chapman & Hall/CRC.

McGill, J. I., & van Ryzin, G. J. (1999). Revenue management: Research overview and prospects. *Transportation Science, 33*(2), 233–256.

Moore, J. M. (1968). An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science, 15*, 102–109.

PG&E (2018). Electric rates: Residential time-of-use. [online]. https://www.pge.com/tariffs/electric.shtml, accessed on November 15, 2018.

Wan, G., & Qi, X. (2010). Scheduling with variable time slot costs. *Naval Research Logistics, 57*, 159–171.

Wan, G., & Yen, B. P. C. (2009). Single machine scheduling to minimize total weighted earliness subject to minimal number of tardy jobs. *European Journal of Operational Research, 195*(1), 89–97.

Wang, Y., Meng, Q., & Du, Y. (2015). Liner container seasonal shipping revenue management. *Transportation Research Part B: Methodological, 82*, 141–161.

Weatherford, L. R., & Bodily, S. E. (1992). A taxonomy and research overview of perishable-asset revenue management: Yield management, overbooking, and pricing. *Operations Research, 40*(5), 831–844.