Ann Oper Res
DOI 10.1007/s10479-012-1294-z

Flow shop scheduling with peak power consumption constraints

Kan Fang · Nelson A. Uhan · Fu Zhao · John W. Sutherland

Abstract We study scheduling as a means to address the increasing energy concerns in manufacturing enterprises. In particular, we consider a flow shop scheduling problem with a restriction on peak power consumption, in addition to the traditional time-based objectives. We investigate both mathematical programming and combinatorial approaches to this scheduling problem, and test our approaches with instances arising from the manufacturing of cast iron plates.

Keywords Scheduling · Flow shop · Energy · Peak power consumption · Integer programming · Combinatorial optimization

1 Introduction

Under the pressures of climate change, increasing energy costs, and growing energy security concerns, manufacturing enterprises have become more and more interested in reducing their energy and power consumption. Until recently, most efforts aimed at minimizing energy and power consumption in manufacturing have been focused on developing machines

K. Fang
School of Industrial Engineering, Purdue University, West Lafayette, IN 47907, USA
e-mail: fang19@purdue.edu

N.A. Uhan (✉)
Mathematics Department, United States Naval Academy, Annapolis, MD 21402, USA
e-mail: uhan@usna.edu

F. Zhao
Division of Environmental and Ecological Engineering and School of Mechanical Engineering, Purdue University, West Lafayette, IN 47904, USA
e-mail: fzhao@purdue.edu

J.W. Sutherland
Division of Environmental and Ecological Engineering, Purdue University, West Lafayette, IN 47907, USA
e-mail: jwsuther@purdue.edu

Published online: 22 January 2013

and equipment that are more energy and power efficient. However, several researchers have observed that in various manufacturing enterprises, energy and power consumption can also be reduced through alternate *operational* strategies, such as the smarter management and scheduling of machine start-up and machine idling (e.g. Dahmus and Gutowski 2004; Gutowski et al. 2005; Drake et al. 2006). In this work, we explore the use of scheduling as a means to reduce the energy and power consumption of manufacturing enterprises.

Research on using scheduling to reduce energy and power consumption in manufacturing is rather sparse. One exception is the work by Mouzon et al. (2007), who proposed several dispatching rules and a multi-objective mathematical programming formulation for scheduling jobs on a single CNC machine in a way that minimizes energy consumption and total completion time. In addition, Mouzon and Yildirim (2008) proposed a metaheuristic algorithm to minimize the total energy consumption and total tardiness on a single machine. Fang et al. (2011) provided some preliminary insights into the modeling and algorithmic challenges of shop scheduling problems with energy and power criteria.

On the other hand, there is a considerable body of literature on scheduling computer processors in a way that minimizes energy consumption. In these settings, the processors can be run at varying speeds: reducing the speed of a processor lowers power consumption, but results in longer processing time. This energy saving technique is called *dynamic voltage scaling*, or *speed scaling*, and was originally studied by Yao et al. (1995). In most of the research on speed scaling, it is assumed that the processor speed can be chosen arbitrarily from a continuous range and the associated power consumption is an exponential function of the speed. One example of work done under this assumption is that of Bansal et al. (2007), who studied the design and performance of speed scaling algorithms for scheduling jobs with deadlines to address concerns with energy and temperature. Other researchers have also considered the case where only a number of discrete speeds are available. For example, Kwon and Kim (2005) proposed a voltage allocation technique for discrete supply voltages to produce a preemptive task schedule that minimizes total energy consumption. Other power functions have also been considered in the literature (e.g. Bansal et al. 2009). For more pointers on the ever-growing literature on speed scaling, we refer the reader to the surveys by Irani and Pruhs (2005) and Albers (2010).

In the speed scaling literature, it is typically assumed that each job needs to be processed on a single processor or one of multiple parallel processors. This is to be expected, as this matches typical computing environments. However, in a typical manufacturing environment, jobs often need to be processed on multiple machines in some order; in other words, in some kind of *job shop environment*. As a result, much of the work on speed scaling is not directly applicable to the problems faced by manufacturing enterprises. In this work, we aim to begin to fill this gap. In particular, we consider a *permutation flow shop scheduling problem* with objectives based on time and power. To the best of our knowledge, our paper is one of the first in-depth studies of shop scheduling with both time and power related criteria.

Most recent research assumes that reducing the average power consumption proportionately decreases energy costs. However, *peak power consumption*—that is, the maximum power consumption over all time instants—also plays a key role in the energy costs of electricity-intensive manufacturing enterprises; for example, many energy providers use time-of-use (TOU) tariffs (e.g. Babu and Ashok 2008). Peak power consumption has also received some attention in the speed scaling literature, since it affects the power supply and cooling technologies in the design of computer processors. Several researchers have studied different approaches to reduce peak power consumption on a single processor or parallel processors (e.g. Felter et al. 2005; Kontorinis et al. 2009; Cochran et al. 2011).

Due to the nature of energy costs in manufacturing environments, we consider the multi-objective problem of minimizing the makespan and the peak power consumption in a permutation flow shop. In particular, we search for Pareto optimal schedules, or schedules for which no other schedule has both lower makespan and lower peak power consumption. In order to handle the bicriteria nature of this problem, we fix an upper bound on the peak power consumption, and minimize the makespan of the schedule. We consider the problem when speeds are discrete and when they are continuous. In addition, we consider flow shops with both zero and unlimited intermediate storage between machines. (In a flow shop with zero intermediate storage, a completed job cannot leave the current machine until the next machine is available.) For simplicity, in this paper, we refer to this problem as the *permutation flow shop scheduling problem with peak power consumption constraints*, or the PFSPP problem for short.

We consider both mathematical programming and combinatorial approaches to the PFSPP problem. Unlike most classical scheduling problems, we need to be able to keep track of which jobs are running concurrently at any time in order to take the peak power consumption into account. This presents some interesting modeling and algorithmic challenges, and some of our results may be of interest in other scheduling applications (e.g. Thörnblad et al. 2010). For the case of discrete speeds and unlimited intermediate storage, we propose two mixed integer programming formulations, inspired by existing formulations for shop scheduling problems (Manne 1960; Lasserre and Queyranne 1992). In order to strengthen these formulations, we give valid inequalities that exploit the structure of optimal schedules and the properties of concurrently running jobs. We also test the computational performance of these two formulations and the effectiveness of these valid inequalities on a set of instances based on the manufacture of cast iron plates with slots.

We also examine the PFSPP problem with two machines and zero intermediate storage. When speeds are discrete, we show that this problem is equivalent to a special case of the asymmetric traveling salesperson problem. We also consider the case when speeds are continuous and the power consumption is an exponential function of speed. In this case, we show that there exists an optimal schedule in which the total peak power consumption at any time instant is equal to exactly the fixed upper bound, and that this problem can be transformed to an equivalent asymmetric traveling salesperson problem. Moreover, if the jobs have some special features, we obtain combinatorial polynomial time algorithms for finding optimal schedules.

2 Mathematical description of the problem

As we mentioned in the introduction, we refer to the flow shop scheduling problem that we study in this paper as the *permutation flow shop problem with power consumption constraints* (or the PFSPP problem for short). An instance of the PFSPP problem consists of a set $\mathcal{J} = \{1, 2, \ldots, n\}$ of jobs and a set $\mathcal{M} = \{1, 2, \ldots, m\}$ of machines. Each job j on machine i has a work requirement p_{ij}, and must be processed nonpreemptively first on machine 1, then on machine 2, and so on. There is a set of speeds $\mathcal{S}$: a job $j \in \mathcal{J}$ processed on machine $i \in \mathcal{M}$ at speed $s \in \mathcal{S}$ has an associated processing time p_{ijs} and power consumption q_{ijs}. In addition, we are given a threshold $Q_{\max}$ on the total power consumption at any time instant of the schedule.

Assumption 2.1 We assume that when we process a job at a higher speed, its processing time decreases, while its power consumption increases. That is, as s increases, p_{ijs} decreases

and q_{ijs} increases. In addition, we assume that the power consumption associated with processing a job on a machine at a particular speed is constant from the job's start time until but not including the job's completion time. Finally, we assume that $\min_{s\in\mathcal{S}}\{q_{ijs}\} \leq Q_{\max}$ for all $i \in \mathcal{M}$ and $j \in \mathcal{J}$.

In this paper, we focus on minimizing the *makespan* $C_{\max}$, the completion time of the last job on the last machine m. We define a *feasible schedule* as a schedule in which the total power consumption at any time is no more than the given threshold $Q_{\max}$. Then, depending on the type of the speed set and flow shop environment, we define the following variants of the PFSPP problem.

Problem 2.2 The set of speeds set $\mathcal{S} = \{s_1, s_2, \ldots, s_d\}$ is discrete. The flow shop has unlimited intermediate storage. Each job $j \in \mathcal{J}$ on machine $i \in \mathcal{M}$ has processing time p_{ijs} and power consumption q_{ijs} when processed at speed $s \in \mathcal{S}$. Find a feasible schedule that minimizes the makespan $C_{\max}$.

Instead of giving an explicit equation for power as a function of speed, we assume that the relationship between processing time, power consumption and speed can be arbitrary in Problem 2.2, as long as it satisfies Assumption 2.1. Without loss of generality, we assume that $s_1 < s_2 < \cdots < s_d$.

We also consider a variant of Problem 2.2, in which the flow shop has two machines and zero intermediate storage.

Problem 2.3 The set of speeds $\mathcal{S} = \{s_1, s_2, \ldots, s_d\}$ is discrete. The flow shop has two machines, that is, $\mathcal{M} = \{1, 2\}$, and zero intermediate storage. Each job $j \in \mathcal{J}$ on machine $i \in \mathcal{M}$ has processing time p_{ijs} and power consumption q_{ijs} when processed at speed $s \in \mathcal{S}$. Find a feasible solution that minimizes the makespan $C_{\max}$.

Unlike in Problems 2.2 and 2.3, it might be the case that each job can run at an arbitrary speed within a given continuous range. It is typical to have power as an exponential function of speed (e.g. Bouzid 2005; Mudge 2001). We consider the following two machine variant of the PFSPP problem with zero intermediate storage.

Problem 2.4 The set of speeds $\mathcal{S} = [s_{\min}, s_{\max}]$ is continuous. The flow shop has two machines, that is, $\mathcal{M} = \{1, 2\}$, and zero intermediate storage. Each job j processed on machine i at speed $s \in \mathcal{S}$ has processing time $p_{ijs} = p_{ij}/s$ and power consumption $q_{ijs} = s^{\alpha}$ for some constant $\alpha > 1$. Find a feasible schedule that minimizes the makespan $C_{\max}$.

3 Discrete speeds and unlimited intermediate storage: mixed integer programming formulations

A great deal of research has focused on solving the ordinary permutation flow shop scheduling problem (without power consumption considerations) with integer programming approaches. These efforts have primarily looked at two families of mixed integer programs. One is based on Manne's (1960) formulation that uses linear ordering variables and pairs of dichotomous constraints (called the *disjunctive constraints*) to ensure one job is processed before another or vice versa. The other is based on Wagner's (1959) use of the classical assignment problem to assign jobs to positions on machines.

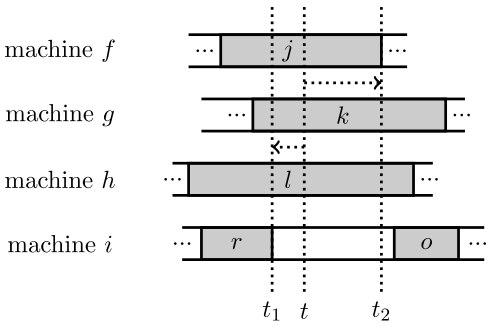


Fig. 1 Gantt chart for Example 3.1

Researchers have investigated the computational performance of different mixed integer programs for the ordinary permutation flow shop scheduling problem from these two families with respect to various objectives (e.g. Stafford et al. 2005; Keha et al. 2009; Unlu and Mason 2010).

In this work, we propose two mixed integer programming formulations, inspired by the work of Manne (1960) and Wagner (1959). In particular, we follow a variant of Wagner's formulation proposed by Lasserre and Queyranne (1992), which models the relationship between jobs and their position in a permutation. We will compare the performance of these two mixed integer programming formulations to discover some promising formulation paradigms that can subsequently be applied to solve larger scheduling problems with peak power consumption constraints.

Unlike most ordinary flow shop scheduling problems, in the PFSPP problem, we need to keep track of jobs that are running concurrently on machines at any time. For this reason, we cannot directly apply the existing mixed integer programming formulations mentioned above. In order to build upon these formulations, we use the following observation. Note that in the PFSPP problem, each job must be processed nonpreemptively with exactly one speed $s \in \mathcal{S}$ on each machine. As a result, when a job is started on a given machine, the power consumption of that machine will stay the same until this job is finished. For any time instance t, let J_t be the set of jobs being processed at t. Let C_t^L (S_t^L) be the completion (start) time of the last job that is completed (started) before time t. Let C_t^R (S_t^R) be the completion (start) time of the first job that is completed (started) after time t. Denote $t_1 = \max\{C_t^L, S_t^L\}$ and $t_2 = \min\{C_t^R, S_t^R\}$. Then for any time t' within $[t_1, t_2)$, we have $J_{t'} = J_t$, which implies that the total power consumption in the flow shop is a constant between t_1 and t_2.

Example 3.1 Consider the following example that illustrates the above observation. At time t, there are three jobs $J_t = \{j, k, l\}$ that are processed concurrently (see Fig. 1). S_t^L is the start time of job k on machine g, C_t^L is the completion time of job r on machine i. In this example, $S_t^L < C_t^L$, so we have $t_1 = C_t^L$. Similarly, we have $t_2 = C_t^R$, which is the completion time of job j on machine f. Then within $[t_1, t_2)$, the total power consumption in the flow shop is constant.

Inspired by this observation, we propose mixed integer programs with binary variables that keep track of jobs that are running concurrently on any two different machines *only at job start and completion times*. First, we propose a mixed integer program in Sect. 3.1 inspired by Manne (1960), which we will call the *disjunctive formulation*. In Sect. 3.2, we propose another mixed integer program inspired by Lasserre and Queyranne (1992), which we will call the *assignment and positional formulation* (or AP formulation for short). For the remainder of this section, we assume that all data are integer.

3.1 Disjunctive formulation

In this section, we propose a mixed integer program inspired by Manne's (1960) formulation. We define the following decision variables:

- $C_{\max}$ is the makespan of the schedule;
- C_{ij} is the completion time of job j on machine i;
- S_{ij} is the start time of job j on machine i;
- δ_{jk} is equal to 1 if job j precedes job k, and 0 otherwise;
- x_{ijs} is equal to 1 if job j is processed on machine i with speed s, and 0 otherwise;
- u_{hkij} is equal to 1 if the start time of job k on machine h is less than or equal to the start time of job j on machine i (in other words, $S_{hk} \leq S_{ij}$), and 0 otherwise;
- v_{hkij} is equal to 1 if the completion time of job k on machine h is greater than the start time of job j on machine i (in other words, $C_{hk} > S_{ij}$), and 0 otherwise;
- y_{hkij} is equal to 1 if the start time of job j on machine i occurs during the processing of job k on machine h (in other words, $S_{hk} \leq S_{ij} < C_{hk}$), and 0 otherwise;
- z_{hksij} is equal to 1 if job k is processed on machine h with speed s, and is processed while job j starts on machine i (in other words, if $x_{hks} = 1$ and $y_{hkij} = 1$), and 0 otherwise.

We call the binary decision variables u, v, y and z the *concurrent job variables*. Let M be an upper bound on the makespan of an optimal schedule. For simplicity, we let $M = \sum_{i\in\mathcal{M}} \sum_{j\in\mathcal{J}} p_{ij1}$. Problem 2.2 can be modeled as follows:

$$\text{minimize} \quad C_{\max} \tag{3.1a}$$

subject to

$$C_{\max} \geq C_{mj} \quad \text{for } j \in \mathcal{J}; \tag{3.1b}$$

$$C_{1j} \geq \sum_{s\in\mathcal{S}} p_{1js} x_{1js} \quad \text{for } j \in \mathcal{J}; \tag{3.1c}$$

$$C_{ij} \geq C_{i-1,j} + \sum_{s\in\mathcal{S}} p_{ijs} x_{ijs} \quad \text{for } i \in \mathcal{M}\backslash\{1\};\ j \in \mathcal{J}; \tag{3.1d}$$

$$C_{ij} \geq C_{ik} + \sum_{s\in\mathcal{S}} p_{ijs} x_{ijs} - M\delta_{jk} \quad \text{for } i \in \mathcal{M};\ j,k \in \mathcal{J},\ k > j; \tag{3.1e}$$

$$C_{ik} \geq C_{ij} + \sum_{s\in\mathcal{S}} p_{iks} x_{iks} - M(1-\delta_{jk}) \quad \text{for } i \in \mathcal{M};\ j,k \in \mathcal{J},\ k > j; \tag{3.1f}$$

$$C_{ij} = S_{ij} + \sum_{s\in\mathcal{S}} p_{ijs} x_{ijs} \quad \text{for } i \in \mathcal{M};\ j \in \mathcal{J}; \tag{3.1g}$$

$$S_{ij} - S_{hk} \leq M u_{hkij} - 1 \quad \text{for } i,h \in \mathcal{M};\ j,k \in \mathcal{J}; \tag{3.1h}$$

$$S_{hk} - S_{ij} \leq M(1-u_{hkij}) \quad \text{for } i,h \in \mathcal{M};\ j,k \in \mathcal{J}; \tag{3.1i}$$

$$C_{hk} - S_{ij} \leq M v_{hkij} \quad \text{for } i,h \in \mathcal{M};\ j,k \in \mathcal{J}; \tag{3.1j}$$

$$S_{ij} - C_{hk} \leq M(1-v_{hkij}) - 1 \quad \text{for } i,h \in \mathcal{M};\ j,k \in \mathcal{J}; \tag{3.1k}$$

$$u_{hkij} + v_{hkij} = 1 + y_{hkij} \quad \text{for } i,h \in \mathcal{M};\ j,k \in \mathcal{J}; \tag{3.1l}$$

$$x_{hks} + y_{hkij} \leq 1 + z_{hksij} \quad \text{for } i,h \in \mathcal{M};\ j,k \in \mathcal{J};\ s \in \mathcal{S}; \tag{3.1m}$$

$$\sum_{s\in\mathcal{S}} x_{ijs} = 1 \quad \text{for } i \in \mathcal{M};\ j \in \mathcal{J}; \tag{3.1n}$$

$$\delta_{jk} + \delta_{kj} = 1 \quad \text{for } j,k \in \mathcal{J},\ j \neq k; \tag{3.1o}$$

$$\delta_{jk} + \delta_{kl} + \delta_{lj} \leq 2 \quad \text{for } j,k,l \in \mathcal{J},\ j \neq k \neq l; \tag{3.1p}$$

$$\sum_{s\in\mathcal{S}} q_{ijs}x_{ijs} + \sum_{h\in\mathcal{M},h\neq i}\sum_{k\in\mathcal{J}}\sum_{s\in\mathcal{S}} q_{hks}z_{hksij} \leq Q_{\max} \quad \text{for } i \in \mathcal{M};\ j \in \mathcal{J}; \tag{3.1q}$$

$$x_{ijs}, \delta_{jk}, u_{hkij}, v_{hkij}, y_{hkij}, z_{hksij} \in \{0,1\} \quad \text{for } i,h \in \mathcal{M};\ j,k \in \mathcal{J};\ s \in \mathcal{S}. \tag{3.1r}$$

The objective (3.1a) and the constraints (3.1b) ensure that the makespan is equal to the completion time of the last job processed on machine m. Constraints (3.1c)–(3.1f) ensure that the completion times are consistent with a flow shop. In particular, constraints (3.1e)–(3.1f) are the disjunctive constraints between any two jobs j and k: they ensure that job j is processed before job k or vice versa. Constraints (3.1g) ensure that jobs are processed nonpreemptively. Constraints (3.1h)–(3.1m) ensure that the concurrent job variables u, v, y and z take their intended values. Constraints (3.1n) indicate that each job can be processed on any given machine with exactly one speed. Constraints (3.1o)–(3.1p) ensure that the jobs are processed in the same order on every machine. Finally, constraints (3.1q) ensure that at any time, the total power consumption across machines does not exceed the threshold $Q_{\max}$.

3.2 Assignment and positional formulation

Next, we give another formulation of our problem, inspired by the formulation proposed by Lasserre and Queyranne (1992), which uses binary variables to directly assign jobs to positions in a permutation. A variant of this formulation was proposed in Fang et al. (2011). We define the following decision variables:

- $C_{\max}$ is the makespan of the schedule;
- C_{ij} is the completion time of the jth job processed on machine i (note that "jth job" refers to the jth position, not job j);
- S_{ij} is the start time of the jth job processed on machine i;
- x_{ijks} is equal to 1 if job j is the kth job processed on machine i with speed s, and 0 otherwise;
- u_{hkij} is equal to 1 if the start time of the kth job processed on machine h is less than or equal to the start time of the jth job processed on machine i (in other words, $S_{hk} \leq S_{ij}$), and 0 otherwise;
- v_{hkij} is equal to 1 if the completion time of the kth job processed on machine h is greater than the start time of the jth job processed on machine i (in other words, $C_{hk} > S_{ij}$), and 0 otherwise;
- y_{hkij} is equal to 1 if the start time of the jth job processed on machine i occurs during the processing of the kth job on machine h (in other words, $S_{hk} \leq S_{ij} < C_{hk}$), and 0 otherwise;
- z_{hlksij} is equal to 1 if job l is the kth job processed on machine h with speed s, and is processed while the jth job starts on machine i (in other words, if $x_{hlks} = 1$ and $y_{hkij} = 1$), and 0 otherwise.

As with the disjunctive formulation, we call the decision variables u, v, y and z the concurrent job variables.

3.2.1 Lower and upper bounds for start and completion time decision variables

For the decision variables representing start and completion times, we can obtain simple lower bounds and upper bounds as follows. Let Ω_{ij} be the set of jobs with the smallest j values of $\{p_{ikd} : k \in \mathcal{J}\}$, and let Δ_j be the set that contains the jobs with the largest j values of $\{\sum_{i\in\mathcal{M}} p_{ik1} : k \in \mathcal{J}\}$. Then, we have the following:

$$S_{ij} \geq \min_{k\in\mathcal{J}}\left\{\sum_{h=1}^{i-1} p_{hkd}\right\} + \sum_{k\in\Omega_{i,j-1}} p_{ikd} \triangleq \underline{S}_{ij} \quad \text{for all } i \in \mathcal{M},\ j \in \{1,\ldots,n\}, \tag{3.2}$$

$$C_{ij} \leq \sum_{k\in\Delta_{j-1}}\sum_{i\in\mathcal{M}} p_{ik1} + \max_{k\in\mathcal{J}\setminus\Delta_{j-1}}\left\{\sum_{h=1}^{i} p_{hk1}\right\} \triangleq \overline{C}_{ij} \quad \text{for all } i \in \mathcal{M},\ j \in \{1,\ldots,n\}. \tag{3.3}$$

Clearly, the upper bound $\overline{C}_{mn}$ for C_{mn} is also an upper bound for the makespan. For simplicity's sake, we define $\overline{S}_{ij} = \overline{C}_{ij}$ and $\underline{C}_{ij} = \underline{S}_{ij}$ for all $i \in \mathcal{M}$ and $j = 1,\ldots,n$.

3.2.2 Basic AP formulation

Using the above lower and upper bounds, we can formulate Problem 2.2 as follows:

$$\text{minimize} \quad C_{\max} \tag{3.4a}$$

subject to

$$C_{\max} \geq C_{mn} \tag{3.4b}$$

$$C_{11} \geq \sum_{j\in\mathcal{J}}\sum_{s\in\mathcal{S}} p_{1js}x_{1j1s} \tag{3.4c}$$

$$C_{ik} \geq C_{i-1,k} + \sum_{j\in\mathcal{J}}\sum_{s\in\mathcal{S}} p_{ijs}x_{ijks} \quad \text{for } i \in \mathcal{M}\setminus\{1\};\ k \in \{1,2,\ldots,n\}; \tag{3.4d}$$

$$C_{ik} \geq C_{i,k-1} + \sum_{j\in\mathcal{J}}\sum_{s\in\mathcal{S}} p_{ijs}x_{ijks} \quad \text{for } i \in \mathcal{M};\ k \in \{2,\ldots,n\}; \tag{3.4e}$$

$$C_{ik} = S_{ik} + \sum_{j\in\mathcal{J}}\sum_{s\in\mathcal{S}} p_{ijs}x_{ijks} \quad \text{for } i \in \mathcal{M};\ k \in \{1,2,\ldots,n\}; \tag{3.4f}$$

$$S_{ij} - S_{hk} \leq (\overline{S}_{ij} - \underline{S}_{hk})u_{hkij} - 1 \quad \text{for } i,h \in \mathcal{M};\ j,k \in \{1,2,\ldots,n\}; \tag{3.4g}$$

$$S_{hk} - S_{ij} \leq (\overline{S}_{hk} - \underline{S}_{ij})(1 - u_{hkij}) \quad \text{for } i,h \in \mathcal{M};\ j,k \in \{1,2,\ldots,n\}; \tag{3.4h}$$

$$C_{hk} - S_{ij} \leq (\overline{C}_{hk} - \underline{S}_{ij})v_{hkij} \quad \text{for } i,h \in \mathcal{M};\ j,k \in \{1,2,\ldots,n\}; \tag{3.4i}$$

$$S_{ij} - C_{hk} \leq (\overline{S}_{ij} - \underline{C}_{hk})(1 - v_{hkij}) - 1 \quad \text{for } i,h \in \mathcal{M};\ j,k \in \{1,2,\ldots,n\}; \tag{3.4j}$$

$$u_{hkij} + v_{hkij} = 1 + y_{hkij} \quad \text{for } i,h \in \mathcal{M};\ j,k \in \{1,2,\ldots,n\}; \tag{3.4k}$$

$$x_{hlks} + y_{hkij} \leq 1 + z_{hlksij} \quad \text{for } i,h \in \mathcal{M};\ j,k,l \in \{1,2,\ldots,n\};\ s \in \mathcal{S}; \tag{3.4l}$$

$$y_{hkij} \leq u_{hkij} \quad \text{for } i,h \in \mathcal{M};\ j,k \in \{1,2,\ldots,n\}; \tag{3.4m}$$

$$y_{hkij} \leq v_{hkij} \quad \text{for } i,h \in \mathcal{M};\ j,k \in \{1,2,\ldots,n\}; \tag{3.4n}$$

$$\sum_{k=1}^{n}\sum_{s\in\mathcal{S}} x_{ijks} = 1 \quad \text{for } i \in \mathcal{M};\ j \in \mathcal{J}; \tag{3.4o}$$

$$\sum_{j\in\mathcal{J}}\sum_{s\in\mathcal{S}} x_{ijks} = 1 \quad \text{for } i \in \mathcal{M};\ k \in \{1,2,\ldots,n\}; \tag{3.4p}$$

$$\sum_{s\in\mathcal{S}} x_{ijks} = \sum_{s\in\mathcal{S}} x_{hjks} \quad \text{for } i,h \in \mathcal{M};\ j,k \in \{1,2,\ldots,n\}; \tag{3.4q}$$

$$\sum_{h\in\mathcal{M},h\neq i}\sum_{l\in\mathcal{J}}\sum_{r=1}^{n}\sum_{s\in\mathcal{S}} q_{hls}z_{hlrsik} + \sum_{j\in\mathcal{J}}\sum_{s\in\mathcal{S}} q_{ijs}x_{ijks} \leq Q_{\max} \quad \text{for } i \in \mathcal{M};\ k \in \{1,2,\ldots,n\}; \tag{3.4r}$$

$$x_{ijks}, u_{hkij}, v_{hkij}, y_{hkij}, z_{hlksij} \in \{0,1\} \quad \text{for } i,h \in \mathcal{M};\ j,l,k \in \{1,2,\ldots,n\};\ s \in \mathcal{S}. \quad (3.4s)$$

The objective (3.4a) and the constraint (3.4b) ensures that the makespan of the schedule is equal to the completion time of the last job on the last machine. Constraints (3.4c)–(3.4e) ensure that the completion times are consistent with a flow shop. Constraints (3.4f) ensure that jobs are processed nonpreemptively. Constraints (3.4g)–(3.4n) ensure that the concurrent job variables u, v, y and z take their intended values. Constraints (3.4o) ensure that on each machine each job is processed with exactly one speed and one position. Constraints (3.4p) ensure that each position is assigned with exactly one job and one speed. Constraints (3.4q) ensure that the jobs are processed in the same order on each machine. Finally, constraints (3.4r) ensure that at any time, the total power consumption across machines is at most $Q_{\max}$. We call the above formulation (3.4a)–(3.4s) the *basic AP formulation*.

3.2.3 Strengthening the basic AP formulation: concurrent job valid inequalities

Recall that the variables u, v, y, z are related to the jobs running concurrently at job start and completion times. In this subsection, we show how to strengthen the basic AP formulation by giving valid inequalities based on the definitions of these variables. We call these inequalities the *concurrent job valid inequalities*.

Theorem 3.2 *The following inequalities are valid for the mixed integer program* (3.4a)–(3.4s):

$$\sum_{r=k+1}^{n} u_{hrij} \le (n-k)u_{hkij} \quad \text{for } i,h \in \mathcal{M};\ j,k \in \{1,2,\ldots,n\}. \quad (3.5a)$$

Proof Fix h,k,i,j. If $u_{hkij} = 0$, i.e. $S_{hk} > S_{ij}$, then for each $r = k+1,\ldots,n$, we must also have $S_{hr} > S_{ij}$ in any feasible schedule, and so by the definition of the variables u, we have $u_{hrij} = 0$. On the other hand, if $u_{hkij} = 1$, the left hand side of inequality (3.5a) is at most $n-k$, since u_{hrij} for $r = k+1,\ldots,n$ are binary variables. Therefore, the above inequality is valid. □

Theorem 3.3 *The following inequalities are valid for the mixed integer program* (3.4a)–(3.4s):

$$\sum_{r=1}^{k-1} v_{hrij} \le (k-1)v_{hkij} \quad \text{for } i,h \in \mathcal{M};\ j,k \in \{1,2,\ldots,n\}. \quad (3.5b)$$

Proof Fix h,k,i,j. If $v_{hkij} = 0$, i.e. $C_{hk} \le S_{ij}$, then for each $r = 1,2,\ldots,k-1$, we must also have $C_{hr} \le S_{ij}$ in any feasible schedule, and so by the definition of variables v, we have $v_{hrij} = 0$. On the other hand, if $v_{hkij} = 1$, the left hand side of inequality (3.5b) is at most $k-1$, since v_{hrij} for $r = 1,2,\ldots,k-1$ are binary variables. Therefore, the above inequality is valid. □

Theorem 3.4 *The following inequalities are valid for the mixed integer program* (3.4a)–(3.4s):

$$y_{hkij} = \sum_{l\in\mathcal{J}} \sum_{s\in\mathcal{S}} z_{hlksij} \quad \text{for } i,h \in \mathcal{M};\ j,k \in \{1,2,\ldots,n\}. \quad (3.5c)$$

Proof Fix h, k, i, j. If $y_{hkij} = 0$, then by the definition of variables z, we have $z_{hlksij} = 0$ for all l and s. Now suppose $y_{hkij} = 1$. From constraints (3.4p) we have $\sum_{l\in\mathcal{J}} \sum_{s\in\mathcal{S}} x_{hlks} = 1$. Without loss of generality, suppose job r is assigned to position k on machine h with speed s; i.e., $x_{hrks} = 1$. Then by the definition of variables z, we have $z_{hrksij} = 1$ and $z_{hlksij} = 0$ for any other job $l \neq r$, and so $\sum_{l\in\mathcal{J}} \sum_{s\in\mathcal{S}} z_{hlksij} = 1$. □

Theorem 3.5 *The following inequalities are valid for the mixed integer program* (3.4a)–(3.4s):

$$\sum_{k=1}^{n} y_{hkij} \leq 1 \quad \textit{for } i, h \in \mathcal{M};\ j \in \{1, 2, \ldots, n\}. \tag{3.5d}$$

Proof For each position j on machine i, there exists at most one position k on machine h that satisfies $S_{hk} \leq S_{ij} < C_{hk}$, since at most one job is processed in each position. □

Theorem 3.6 *The following inequalities are valid for the mixed integer program* (3.4a)–(3.4s):

$$\begin{aligned} &u_{hkij} + u_{ijhk} \\ &\quad \leq \frac{1}{2}(y_{hkij} + y_{ijhk}) + 1 \leq v_{hkij} + v_{ijhk} \quad \textit{for } i, h \in \mathcal{M};\ j, k \in \{1, 2, \ldots, n\}. \end{aligned} \tag{3.5e}$$

Proof Fix h, k, i, j. Suppose $y_{hkij} + y_{ijhk} = 0$, i.e., $y_{hkij} = y_{ijhk} = 0$, or $S_{hk} \geq C_{ij}$ or $S_{ij} \geq C_{hk}$. Without loss of generality, assume $S_{hk} \geq C_{ij}$, which implies $C_{hk} \geq S_{hk} \geq C_{ij} \geq S_{ij}$. Then, by definition we have $u_{hkij} = 0$, $u_{ijhk} = 1$, $v_{hkij} = 1$, and $v_{ijhk} = 0$, and so the inequality (3.5e) holds. Now suppose $y_{hkij} + y_{ijhk} = 1$. Without loss of generality, assume $y_{hkij} = 1$ and $y_{ijhk} = 0$. Then we have $S_{hk} \leq S_{ij} < C_{hk}$. By definition we have $u_{hkij} = 1$, $u_{ijhk} = 0$, and $v_{hkij} = v_{ijhk} = 1$, and so the inequality (3.5e) also holds. Finally, if $y_{hkij} + y_{ijhk} = 2$, i.e., $y_{hkij} = y_{ijhk} = 1$, or $S_{hk} = S_{ij}$, then by definition we have $u_{hkij} = u_{ijhk} = 1$ and $v_{hkij} = v_{ijhk} = 1$, and so inequality (3.5e) still holds. □

Theorem 3.7 *For each $i, h \in \mathcal{M}$ and $j \in \{1, 2, \ldots, n\}$, the following inequalities are valid for the mixed integer program* (3.4a)–(3.4s):

$$u_{h,k+1,ij} + v_{h,k-1,ij} \leq 1 - y_{hkij} \quad \textit{for } k \in \{2, \ldots, n-1\}. \tag{3.5f}$$

Proof Fix h, k, i, j. If $y_{hkij} = 1$, i.e., $S_{hk} \leq S_{ij} < C_{hk}$, then we have $S_{h,k+1} > S_{ij}$ and $C_{h,k-1} \leq S_{ij}$, or in other words, $u_{h,k+1,ij} = v_{h,k-1,ij} = 0$. So in this case, inequality (3.5f) holds. Otherwise, because $C_{h,k-1} < S_{h,k+1}$, we have that $S_{h,k+1} \leq S_{ij}$ and $C_{h,k-1} > S_{ij}$ cannot be satisfied simultaneously, or in other words, $u_{h,k+1,ij} + v_{h,k-1,ij} \leq 1$. So in this case as well, inequality (3.5f) holds. □

Theorem 3.8 *For each $i, h \in \mathcal{M}$ and $j \in \{2, \ldots, n-1\}$, the following inequalities are valid for the mixed integer program* (3.4a)–(3.4s):

$$u_{hki,j+1} + v_{hki,j-1} \geq 1 + y_{hkij} \quad \textit{for } k \in \{1, 2, \ldots, n\}. \tag{3.5g}$$

Proof Fix h, k, i, j. If $y_{hkij} = 1$, i.e., $S_{hk} \leq S_{ij} < C_{hk}$, then we have $S_{hk} < S_{i,j+1}$ and $C_{hk} > S_{i,j-1}$. In other words, $u_{hki,j+1} = v_{hki,j-1} = 1$, and so in this case, inequality (3.5g) holds. Otherwise, we have that either $C_{i,j-1} \leq S_{hk}$ or $C_{hk} \leq S_{i,j+1}$, which implies $u_{hki,j+1} + v_{hki,j-1} \geq 1$. So in this case as well, inequality (3.5g) holds. □

3.2.4 Strengthening the basic AP formulation: nondelay valid inequalities

A feasible schedule is called *nondelay* if no machine is idle when there exists a job that can be processed without violating the threshold $Q_{\max}$ on the total power consumption across all machines.

Theorem 3.9 *For Problem 2.2, there always exists an optimal schedule that is nondelay.*

Proof Suppose schedule σ_1 is an optimal schedule that is not nondelay for Problem 2.2. Let C_{σ_1} be the makespan of σ_1. Suppose machine i is idle when job j is available to be processed in schedule σ_1. Then we process all the other jobs the same way, and process job j with the same speed as in schedule σ_1, starting at the earliest time after $C_{i-1,j}$ at which scheduling job j on machine i does not violate the peak power consumption constraints. Denote this new schedule as σ_2 with makespan C_{σ_2}. If the completion time of job j on machine i in σ_1 is not the unique value that determines the makespan, then we still have $C_{\sigma_2} = C_{\sigma_1}$. Now suppose j is the only job that attains the makespan C_{σ_1} in schedule σ_1. Then by processing job j earlier on machine i, we obtain $C_{\sigma_2} < C_{\sigma_1}$, which contradicts the optimality of schedule σ_1. □

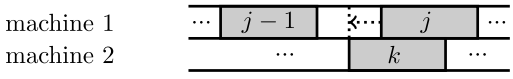
Based on Theorem 3.9, we can add constraints to the basic AP formulation (3.4a)–(3.4s) that require schedules to be nondelay. It is difficult to obtain such inequalities for the general PFSPP problem. However, when the flow shop has two machines, i.e., $\mathcal{M} = \{1, 2\}$, then we have the following *nondelay valid inequalities* (3.6a)–(3.6e).

Theorem 3.10 *Suppose $\mathcal{M} = \{1, 2\}$. For each $j \in \{2, 3, \ldots, n\}$, the following inequalities are valid for the mixed integer program* (3.4a)–(3.4s):

$$y_{2k1j} + 1 - v_{1,j-1,2k} \le u_{2k1j} + u_{1j2k} \quad \text{for } k \in \{1, 2, \ldots, j-1\}, \tag{3.6a}$$

$$y_{1k2j} + 1 - v_{2,j-1,1k} \le u_{1k2j} + u_{2j1k} \quad \text{for } k \in \{j+1, \ldots, n\}. \tag{3.6b}$$

Proof Fix j, k. If $y_{2k1j} = 1$, i.e., $S_{2k} \le S_{1j} < C_{2k}$, and $v_{1,j-1,2k} = 0$, i.e., $C_{1,j-1} \le S_{2k}$, then because there always exists an optimal nondelay schedule, we can start the job in the jth position on machine 1 simultaneously with the job in the kth position on machine 2, i.e., $S_{2k} = S_{1j}$.



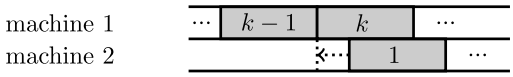
In other words, $u_{2k1j} = u_{1j2k} = 1$. So in this case the inequality (3.6a) holds. Otherwise, if $y_{2k1j} = 0$, because $u_{2k1j} + u_{1j2k} \ge 1$ and $v_{1,j-1,2k} \ge 0$, the inequality (3.6a) also holds.

Reversing the roles of machines 1 and 2, we similarly obtain valid inequalities (3.6b). □

Theorem 3.11 *Suppose $\mathcal{M} = \{1, 2\}$. The following inequalities are valid for the mixed integer program* (3.4a)–(3.4s):

$$y_{1k21} \le u_{211k} \quad \text{for } k \in \{2, 3, \ldots, n\}. \tag{3.6c}$$

Proof Fix k. If $y_{1k21} = 1$, i.e. $S_{1k} \le S_{21} < C_{1k}$, then we can start the job in the first position on machine 2 simultaneously with the job in the kth position on machine 1, that is $S_{1k} = S_{21}$.



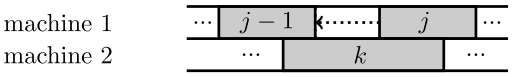
In other words, $u_{211k} = 1$, and so inequality (3.6c) holds. □

Theorem 3.12 *Suppose* $\mathcal{M} = \{1, 2\}$. *For each* $j \in \{2, 3, \ldots, n\}$, *the following inequalities are valid for the mixed integer program* (3.4a)–(3.4s):

$$S_{1j} - C_{1,j-1} \le (\overline{S}_{1j} - \underline{C}_{1,j-1})(2 - v_{1,j-1,2k} - y_{2k1j}) \quad \text{for } k \in \{1, 2, \ldots, j-2\}, \tag{3.6d}$$

$$S_{2j} - C_{2,j-1} \le (\overline{S}_{2j} - \underline{C}_{2,j-1})(2 - v_{2,j-1,1k} - y_{1k2j}) \quad \text{for } k \in \{j+1, \ldots, n\}. \tag{3.6e}$$

Proof Fix j, k. Suppose $y_{2k1j} = 1$ and $v_{1,j-1,2k} = 1$, i.e., $S_{2k} \le S_{1j} < C_{2k}$ and $C_{1,j-1} > S_{2k}$.



Then because there always exists an optimal nondelay schedule, we can process the job in the jth position immediately after the completion time of the job in the $(j-1)$th position on machine 1; that is, $S_{1j} = C_{1,j-1}$. So the inequality (3.6d) holds. Now suppose y_{2k1j} and $v_{1,j-1,2k}$ are not both equal to 1. Then the right side is at least $\overline{S}_{1j} - \underline{C}_{1,j-1}$, and so the inequality (3.6d) still holds.

Reversing the roles of machines 1 and 2, we similarly obtain valid inequalities (3.6e). □

We call the formulation that combines the basic AP formulation with the concurrent job valid inequalities (3.5a)–(3.5g) (and the nondelay valid inequalities (3.6) when $\mathcal{M} = \{1, 2\}$) the *enhanced AP formulation*.

3.3 Experimental study

3.3.1 Computational environment

In this section, we compare the performance of the different formulations presented in Sect. 3.1 and Sect. 3.2, with respect to both computation time and solution quality. We used Gurobi Optimizer 4.5 to solve the mixed integer programs on a computer with two 2.5 GHz Quad-Core AMD 2380 processors and 32 GB of RAM running the Linux operating system.

To conduct our experiments, we considered a hypothetical flow shop scheduling problem arising from the manufacture cast iron plates with slots (Fig. 2). The plates manufactured in this flow shop can have three different lengths, two different depths of milling on the surface, three different numbers of slots, and two different depths of slots. In other words, there are $3 \times 2 \times 3 \times 2 = 36$ different types of plates. There are two types of machines with different operations: face milling and profile milling. We consider two different cases: in the first case, each plate must have the two operations on one side; in the second case, each plate must have the two operations on two sides. We can view these two different cases as two different flow shop problems with 2 and 4 machines, respectively; that is, $\mathcal{M} = \{1, 2\}$ or $\mathcal{M} = \{1, 2, 3, 4\}$. There are five available cutting speeds on each machine; that is, $d = 5$ and $\mathcal{S} = \{s_1, s_2, s_3, s_4, s_5\}$. We also consider a special case in which we can only use each machine's minimum and maximum speeds; that is, $d = 2$ and $\mathcal{S} = \{s_1, s_2\}$. The

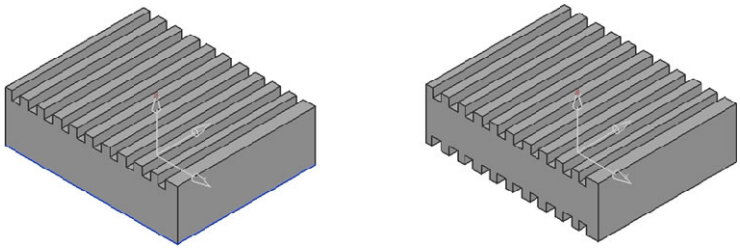


Fig. 2 Cast iron plates with slots

processing times p_{ijs} and power consumption values q_{ijs} are generated in accordance with the Machinery's Handbook (Oberg et al. 2008). In the instances we study, the processing times p_{ijs} are in the interval [4, 22] (in minutes), and the power consumption values q_{ijs} are in the interval [4, 9] (in kW). For this study, we consider instances in which the number of jobs n is 8, 12, 16, or 20.

We generate 10 different settings for each combination of (m, n, d), by randomly sampling n jobs with replacement from the 36 job types. Let (m, n, d, k) denote the kth setting that has m machines, n jobs and d speeds. In summary, the family of settings we use in our experiment is

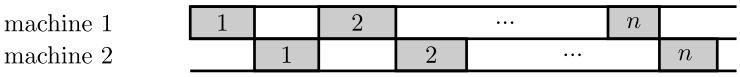
$$\big\{(2, n, 2, k), (2, n, 5, k), (4, n, 2, k) : n \in \{8, 12, 16, 20\},\ k \in \{1, 2, \ldots, 10\}\big\}.$$

For each setting (m, n, d, k), we define $\underline{Q} = \max_{i\in\mathcal{M}, j\in\mathcal{J}}\{q_{ij1}\}$ and $\overline{Q} = \sum_{i\in\mathcal{M}} \max_{j\in\mathcal{J}}\{q_{ijd}\}$. Note that when $Q_{\max} < \underline{Q}$, there is no feasible schedule, and when $Q_{\max} > \overline{Q}$, all jobs can be processed concurrently at their maximum speed. We call $[\underline{Q}, \overline{Q}]$ the *power consumption range* for each instance, which we divide into 9 subintervals with same length. For each setting (m, n, d, k), we solve the corresponding mixed integer program using the 10 endpoints of these subintervals as the threshold $Q_{\max}$ on the total power consumption at any time instant. This way, for each combination of m, n and d, we will test 10 settings with 10 different peak power consumption thresholds, or 100 instances of the problem. We set a 30 minute time limit on each instance. For each instance, we provide Gurobi Optimizer with an upper bound on the optimal makespan, using the best solution obtained from the two heuristic algorithms described next.

3.3.2 Two heuristic algorithms for finding feasible schedules

For an instance (m, n, d, k) with threshold $Q_{\max}$ on the peak power consumption, let s^*_{ij} be the maximum possible speed at which job j can be processed on machine i individually without violating the power consumption threshold; i.e., $s^*_{ij} = \max\{s \in \mathcal{S} : q_{ijs} \le Q_{\max}\}$. Suppose we fix a permutation of the job set. Then one straightforward algorithm (Algorithm 3.1) for finding a feasible schedule is to process each job j on machine i at speed s^*_{ij} without overlap—that is, with no other concurrently running jobs.

For example, when Algorithm 3.1 is applied to a two-machine flow shop, in which each job is processed with its maximum possible speed without overlap, the Gantt chart of the resulting schedule looks like this:



Algorithm 3.1 Maximum possible speed, fix permutation, no overlap

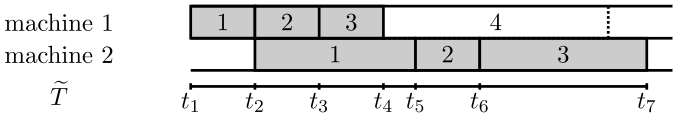
Given p_{ijs}, q_{ijs} for $i \in \mathcal{M}$, $j \in \mathcal{J}$, $s \in \mathcal{S}$, permutation $(1, \ldots, n)$ of $\mathcal{J}$
for $j = 1$ to n **do**
  for $i = 1$ to m **do**
    calculate s^*_{ij}
    schedule job j on machine i with s^*_{ij} without overlap
  end for
end for

Algorithm 3.2 Minimum possible speed, fix permutation, as early as possible

Given p_{ijs}, q_{ijs} for $i \in \mathcal{M}$, $j \in \mathcal{J}$, $s \in \mathcal{S}$, permutation $(1, \ldots, n)$ of $\mathcal{J}$
for $j = 1$ to n **do**
  for $i = 1$ to m **do**
    schedule job j on machine i at its earliest possible time $\widetilde{T}[l]$
    while power consumption constraints are violated **do**
      $l = l + 1$
      schedule j on machine i at the next available time $\widetilde{T}[l]$
    end while
    update $\widetilde{T}$
  end for
end for

Another simple method to generate feasible solutions is to process jobs as early as possible at their minimum possible speeds, while respecting the power consumption constraints (Algorithm 3.2). As mentioned above (e.g., Example 3.1), we only need to keep track of the jobs that are running concurrently at start or completion times. Let $\widetilde{T}$ be the sorted list of all the start and completion times of jobs, and let $\widetilde{T}[i]$ be the ith component in $\widetilde{T}$.

For example, suppose that the following is a Gantt chart for the schedule we obtain using Algorithm 3.2 for the first 3 jobs of a two-machine instance:



Here, $\widetilde{T} = \{t_1, \ldots, t_7\}$. Next, Algorithm 3.2 attempts to schedule job 4 on machine 1 at its earliest possible time t_4. If the power consumption threshold is violated when job 4 is processed on machine 1 at time t_4, then the algorithm tries to schedule job 4 on machine 1 at the next possible time t_5. If the power consumption threshold is satisfied, then the algorithm schedules job 4 on machine 1 at t_5 and updates the list of times $\widetilde{T}$.

The primary role of the two heuristic algorithms described above in our study is to quickly obtain an upper bound on the optimal makespan for use in Gurobi's branch-and-cut procedure. Unfortunately, in our computational results, we found that these upper bounds often are quite loose, since they only consider processing the jobs at the minimum and maximum speeds available. When jobs can be processed at a variety of speeds, the schedules found by the heuristic algorithms usually do not compare well to optimal schedules.

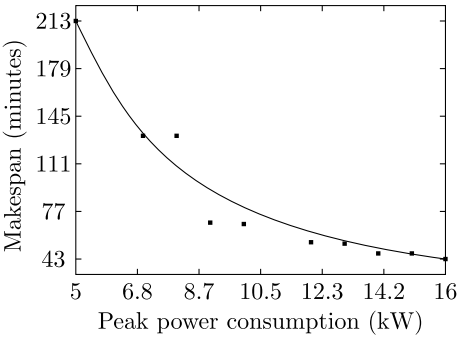


Fig. 3 Trade-off between makespan and peak power consumption

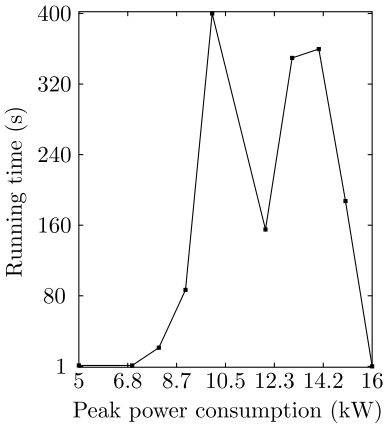


Fig. 4 Relationship between running time and peak power consumption

3.3.3 Experiment 1: makespan ($C_{\max}$) vs. power consumption ($Q_{\max}$)

From Assumption 2.1, we know that when a job is processed at a higher speed, its processing time decreases, while its power consumption increases. Based on this assumption, one expects a significant trade-off between makespan and peak power consumption. To achieve the shortest possible makespan, jobs should be processed at their highest possible speeds simultaneously without idle time, which leads to high peak power consumption. On the other hand, if the objective is to minimize peak power consumption, jobs should be processed at their lowest speeds and the machines should be operated without overlap. Figure 3 shows an example of the relationship between makespan and power consumption for a setting with $m = 2, n = 8$, and $d = 2$.

We also observe that the running times of all the formulations are considerably higher for instances with intermediate values of $Q_{\max}$. Figure 4 shows an example of the relationship between $Q_{\max}$ and running time for the same setting with $m = 2$, $n = 8$ and $d = 2$, using the enhanced AP formulation. This seems correct, intuitively: for extreme values of $Q_{\max}$, the optimal schedule is relatively straightforward to compute, whereas for intermediate values of $Q_{\max}$, the scheduling (in particular, deciding the processing speeds) is trickier.

As we will see in the subsequent subsections, this pattern is prevalent. Since the running times of the formulations appear to be related to the value of $Q_{\max}$, we divide the power consumption range into 3 classes. For an instance with power consumption range $[\underline{Q}, \overline{Q}]$, if $Q_{\max}$ is less than the first quartile of $[\underline{Q}, \overline{Q}]$, then we call $Q_{\max}$ "low"; when $Q_{\max}$ is

Table 1 Comparison of average running time between disjunctive and basic AP formulations

m	d	$Q_{\max}$ range	Formulation	Average running time (s)				
				$n=4$	$n=5$	$n=6$	$n=7$	$n=8$
2	2	low	disjunctive	0.07	0.78	11.81	280.60	NA
			basic AP	0.05	0.15	0.45	0.85	1.12
		intermediate	disjunctive	0.12	0.51	5.87	127.80	NA
			basic AP	0.18	0.31	2.09	17.96	NA
		high	disjunctive	0.09	0.16	0.49	2.83	24.79
			basic AP	0.07	0.06	0.11	0.28	0.62

NA: not all instances solved within the time limit

greater than the third quartile of $[\underline{Q}, \overline{Q}]$, then we call $Q_{\max}$ "high"; otherwise, we call $Q_{\max}$ "intermediate." In the following experiments, we will analyze the performance of the mixed integer programs with respect to different classes of $Q_{\max}$.

3.3.4 Experiment 2: disjunctive vs. basic AP formulation

In order to compare the performance of the different formulations, we use the following measures to assess their performance:

- The number of instances solved to optimality within the 30 minute time limit.
- The average and maximum solution time for these instances solved to optimality.
- The average and maximum *speedup factor* of the running time for these instances solved to optimality. For any two formulations a and b, we define the speedup factor (SF) between a and b for a given instance as the ratio between the times taken for a and b to solve the instance. We only compute a speedup factor when both formulations can solve the instances to optimality within the predetermined time limit.
- The average optimality gap at various time points within the 30 minute time limit. We define the *optimality gap* as the ratio of the value of the best known feasible solution to the best known lower bound. This measure lets us compare the performance of the different formulations for the instances that do not solve to optimality within the time limit.

These performance measures were also used by Keha et al. (2009) and Unlu and Mason (2010) in their study of mixed integer programming formulations for various scheduling problems.

In this experiment, we compare the performance of the disjunctive and basic AP formulations. We look at a family of instances similarly constructed to the one described in Sect. 3.3.1, except that we look at settings with $m = 2$, $n \in \{4, 5, 6, 7, 8\}$, and $d = 2$. We focus on these smaller instances in this experiment because as we see in Table 1, the disjunctive formulation for even moderately sized instances (e.g., $n = 8$) fails to solve within the 30 minute time limit. Table 1 shows the average running time for the disjunctive and basic AP formulations, and Table 2 shows the average speedup factor between the disjunctive and basic AP formulations.

From Tables 1 and 2, we can see that the basic AP formulation runs much faster than the disjunctive formulation, especially for instances with larger numbers of jobs. Figure 5

Table 2 Speedup factor between disjunctive and basic AP formulations

m	d	$Q_{\max}$ range	Average speedup factor				
			$n=4$	$n=5$	$n=6$	$n=7$	$n=8$
2	2	low	1.43	3.63	33.95	346.00	NA
		intermediate	0.76	2.85	16.00	138.00	NA
		high	2.57	4.03	4.95	12.03	52.27

NA: not all instances solved within the time limit

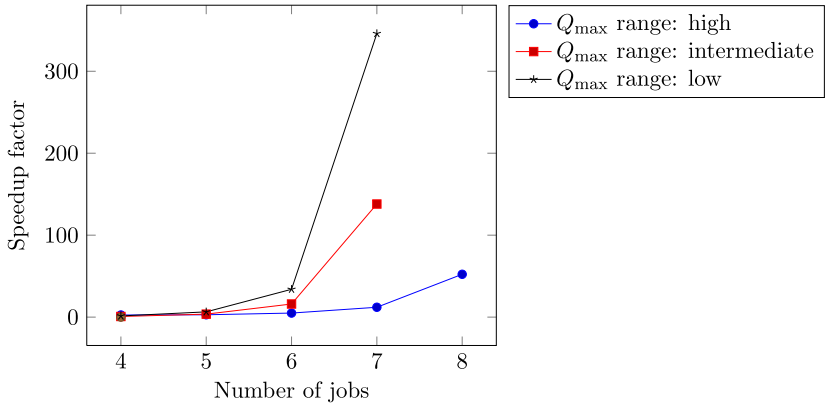


Fig. 5 Average speedup factors between the disjunctive and basic AP formulations

graphically shows the average speedup factor between the disjunctive and basic AP formulations. Given these observations, we decided to devote more attention to strengthened versions of the basic AP formulation.

3.3.5 Experiment 3: basic AP formulation vs. enhanced AP formulation

We proposed the concurrent valid inequalities for basic AP formulation in Sect. 3.2.3, and the nondelay valid inequalities when $m = 2$ in Sect. 3.2.4. We expect that the addition of these inequalities in the enhanced AP formulation will result in better computational results than the basic AP formulation. As mentioned in Sect. 3.3.1, we consider instances with 2 machines and 4 machines.

Table 3 displays the size of the basic and the enhanced AP formulations for an instance with $m = 2$ and $d = 2$. The enhanced AP formulation has the same number of variables as the basic AP formulation, but the enhanced AP formulation has more constraints and nonzeros than the basic AP formulation, since the concurrent job valid inequalities (and nondelay valid inequalities when $m = 2$) are also included.

Table 5 shows the computational results for the instances solved to optimality with low or high values of $Q_{\max}$. From this table, we see that the number of instances solved in the basic AP formulation is about the same as the enhanced AP formulation, except for instances with $m = 4$, $n = 12$, and $d = 2$, for which only one of the instances was solved to optimality using the basic AP formulation while 3 instances were solved using the enhanced AP formulation. We also see that the average speedup factor in many cases is less than 1, meaning that

Table 3 Initial sizes of the basic and the enhanced AP formulations

n	AP formulation	# of variables	# of constraints	# of nonzeros
8	basic	2721	2874	12001
	enhanced	2721	4204	18427
12	basic	8401	8726	36237
	enhanced	8401	11776	54415
16	basic	19009	19570	80825
	enhanced	19009	25044	119523
20	basic	36081	36942	151909
	enhanced	36081	45544	222199

the basic AP formulation solved faster on average than the enhanced AP formulation. This observation might be explained by the low and high $Q_{\max}$: in Sect. 3.3.3 we observed that the instances with low or high values of $Q_{\max}$ are "easy." Given this, one might expect that the running times of these formulations for these instances are mainly based on the size of the formulations, not the scheduling decisions.

Table 4 shows the computational results for the instances with intermediate values of $Q_{\max}$. From this table, we see that in most cases, the average speedup factor is larger than 1. In other words, for these instances with intermediate values of $Q_{\max}$, the additional valid inequalities in the enhanced AP formulation help in reducing running times.

When analyzing the data in more detail, we found more evidence that the additional valid inequalities are effective in reducing running times for instances with intermediate $Q_{\max}$. Table 6 shows the computational results for instances with intermediate values of $Q_{\max}$ in which $m = 2$, $n = 8$, and $d = 2$. From Table 6, we see that for most instances, the enhanced AP formulation runs faster than the basic AP formulation (i.e. 53 out of 60). On the other hand, for instances in which the basic AP formulation is faster, we see that the average running times for both formulations are much smaller (less than 2 seconds).

These observations suggest something similar to what we observed with the data in Table 5. When the problem is easy to solve, the size of formulation is the main factor in the running time, implying that the basic AP formulation should be faster. Otherwise, when the problem is more difficult, the valid inequalities added in the enhanced AP formulation significantly reduce the running time.

We also observed that the majority of instances, especially those with larger m, n, and d, did not solve within 30 minutes, even using the enhanced AP formulation. Tables 7, 8 and 9 show the optimality gap at various times for those instances with both AP formulations within the 30 minute time limit. From Tables 7, 8 and 9, we see that the valid inequalities help in reducing the optimality gap of the mixed integer program at various time points in the solution process (especially at the later times). In addition, we see that as the number of jobs increases, the solution quality decreases at any given time, since the increased number of jobs adds to the difficulty of scheduling jobs.

4 Two machines, discrete speeds, and zero intermediate storage

Recall that in Problem 2.3, we are given a discrete speed set $\mathcal{S} = \{s_1, \ldots, s_d\}$ and a set of two machines $\mathcal{M} = \{1, 2\}$, and there is zero intermediate storage in the flow shop. We will show that this variant of the PFSPP problem can be transformed into an instance of the

Table 4 Results for solved instances with intermediate $Q_{\max}$

Instance type					Basic AP			Enhanced AP			Avg SF	Max SF
m	d	$Q_{\max}$ range	# of instances	n	# solved	Avg time	Max time	# solved	Avg time	Max time		
2	2	intermediate	60	8	59	442.99	1682.51	60	121.41	533.91	4.06	8.66
				12	9	103.63	497.80	9	66.16	210.29	1.87	7.66
				16	8	204.69	1172.95	9	304.31	1452.12	1.57	7.34
				20	8	355.16	889.05	7	440.15	1311.67	0.81	1.74
2	5	intermediate	60	8	38	270.23	1353.97	60	390.02	1591.26	2.11	6.90
				12	17	107.08	226.56	18	207.02	1307.46	0.75	1.57
				16	16	329.35	1697.98	16	373.06	1314.90	0.83	1.64
				20	5	602.92	800.10	5	810.61	1629.28	1.07	2.28
4	2	intermediate	60	8	6	550.60	851.57	9	811.42	1660.00	1.17	1.73
				12	0	–	–	0	–	–	–	–
				16	0	–	–	0	–	–	–	–
				20	0	–	–	0	–	–	–	–

Table 5 Results for solved instances with low or high $Q_{\max}$

Instance type					Basic AP			Enhanced AP			Avg SF	Max SF
m	d	$Q_{\max}$ range	# of instances	n	# solved	Avg time	Max time	# solved	Avg time	Max time		
2	2	low	20	8	20	1.10	1.77	20	1.53	2.90	0.81	1.62
				12	20	109.70	208.30	20	54.67	88.64	1.96	3.68
				16	0	–	–	0	–	–	–	–
				20	0	–	–	0	–	–	–	–
		high	20	8	20	74.81	763.14	20	25.25	300.69	0.93	4.63
				12	19	29.98	223.78	19	48.90	409.30	0.60	2.98
				16	18	118.32	1334.63	18	134.24	792.74	0.51	1.68
				20	18	181.11	1200.92	17	297.88	1313.01	0.53	3.34
2	5	low	20	8	20	1.64	2.50	20	2.00	3.89	0.89	1.70
				12	20	160.55	205.62	20	112.15	386.55	1.67	2.77
				16	0	–	–	0	–	–	–	–
				20	0	–	–	0	–	–	–	–
		high	20	8	20	1.58	11.20	20	1.76	3.51	0.83	3.19
				12	20	8.71	59.62	20	24.05	55.97	0.41	1.07
				16	20	19.56	81.00	20	80.74	213.36	0.40	1.78
				20	20	99.79	514.15	20	249.40	723.71	0.56	4.10
4	2	low	20	8	0	–	–	0	–	–	–	–
				12	0	–	–	0	–	–	–	–
				16	0	–	–	0	–	–	–	–
				20	0	–	–	0	–	–	–	–
		high	20	8	19	406.58	1109.00	19	499.11	1151.89	1.02	4.44
				12	1	1490.82	1490.82	3	1425.87	1532.59	1.20	1.37
				16	0	–	–	0	–	–	–	–
				20	0	–	–	0	–	–	–	–

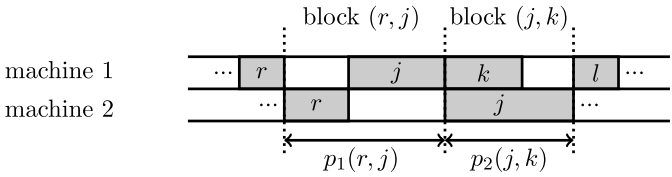
Table 6 Results for instances with intermediate values of $Q_{\max}$ and $m = 2$, $n = 8$, $d = 2$

Result type	# of instances	Formulation	Avg time	Max time	Avg SF	Max SF
Enhanced AP is faster	53	basic AP	527.00	1800.00	4.54	8.66
		enhanced AP	137.23	533.90		
Basic AP is faster	7	basic AP	0.76	1.54	0.46	0.82
		enhanced AP	1.57	2.23		

asymmetric traveling salesperson problem (TSP). Recall that in the asymmetric TSP, we are given a complete directed graph and arc distances, and the task is to find a shortest tour in the graph. This transformation is inspired by a similar transformation of the classic permutation flow shop problem with zero intermediate storage by Reddi and Ramamoorthy (1972).

Before we describe this transformation, we need to establish the notion of a "block" in a schedule for a flow shop with no intermediate storage. Consider the following example.

Example 4.1 In a two machine flow shop with zero intermediate storage, suppose job r, j, k, l are processed successively in a schedule. For example, consider the following Gantt chart.



Note that in this example, because there is zero intermediate storage, job k cannot leave machine 1 until job j is completed on machine 2, and so job l cannot be started on machine 1 until the completion time of job j. We can view any feasible schedule for Problem 2.3 as the combination of $n + 1$ *blocks*: a block (j, k) is a subsequence such that job j is processed on machine 2, and job k is processed on machine 1. We can process each block with overlap (e.g. block (j, k) in the Gantt chart above), or without overlap (e.g. block (r, j)). Moreover, when the permutation of the jobs is fixed, we only need to minimize the total processing time of each block in order to find an optimal schedule.

For any feasible schedule, we define the following quantities. Let $p(j, k)$ denote the minimum total processing time of block (j, k). Let $p_1(j, k)$ be the minimum total processing time of block (j, k) when it is processed without overlap while respecting the power consumption threshold, and let $p_2(j, k)$ be the minimum total processing time of block (j, k) when jobs j and k are processed with overlap while respecting the power consumption threshold. Recall that s^*_{ij} is the maximum speed at which job j can be processed on machine i individually without violating the power consumption threshold. Then, it is straightforward to see that the following holds.

Lemma 4.2

(a) *There exists an optimal schedule for Problem 2.3 such that for each block (j, k) of jobs, we have that $p_1(j, k) = p_{2js^*_{2j}} + p_{1ks^*_{1k}}$ and $p_2(j, k) = \min\{\max\{p_{2js_{2j}}, p_{1ks_{1k}}\} : q_{2js_{2j}} + q_{1ks_{1k}} \le Q_{\max}, s_{2j} \in \mathcal{S}, s_{1k} \in \mathcal{S}\}$.*

Table 7 Average optimality gap for instances with $m = 2$, $d = 2$

Instance type				AP formulations	Average optimality gap at time t (s)								
m	d	$Q_{\max}$ range	n		5	10	30	60	150	300	600	1200	1800
2	2	low	8	basic	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
				enhanced	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
			12	basic	1.39	1.31	1.24	1.12	1.02	1.00	1.00	1.00	1.00
				enhanced	1.38	1.28	1.17	1.04	1.00	1.00	1.00	1.00	1.00
			16	basic	1.61	1.56	1.48	1.47	1.41	1.36	1.31	1.27	1.24
				enhanced	1.60	1.54	1.46	1.43	1.36	1.31	1.27	1.22	1.20
			20	basic	1.65	1.63	1.60	1.57	1.55	1.53	1.50	1.47	1.46
				enhanced	1.65	1.63	1.59	1.56	1.53	1.50	1.47	1.44	1.43
		intermediate	8	basic	1.38	1.34	1.25	1.19	1.11	1.06	1.02	1.01	1.00
				enhanced	1.32	1.25	1.13	1.08	1.03	1.01	1.00	1.00	1.00
			12	basic	1.55	1.46	1.43	1.40	1.36	1.34	1.30	1.28	1.27
				enhanced	1.64	1.47	1.41	1.36	1.31	1.28	1.25	1.21	1.20
			16	basic	1.66	1.65	1.50	1.48	1.43	1.40	1.38	1.37	1.37
				enhanced	1.68	1.67	1.51	1.50	1.42	1.38	1.36	1.34	1.33
			20	basic	1.70	1.70	1.67	1.52	1.49	1.45	1.43	1.41	1.40
				enhanced	1.70	1.70	1.70	1.59	1.54	1.45	1.42	1.40	1.39
		high	8	basic	1.03	1.02	1.01	1.01	1.01	1.01	1.01	1.00	1.00
				enhanced	1.02	1.02	1.01	1.01	1.00	1.00	1.00	1.00	1.00
			12	basic	1.08	1.06	1.03	1.02	1.02	1.01	1.01	1.01	1.01
				enhanced	1.30	1.09	1.06	1.02	1.02	1.01	1.01	1.01	1.01
			16	basic	1.40	1.28	1.09	1.05	1.02	1.02	1.02	1.01	1.01
				enhanced	1.64	1.54	1.19	1.45	1.03	1.02	1.01	1.01	1.01
			20	basic	1.69	1.62	1.29	1.11	1.05	1.02	1.01	1.01	1.00
				enhanced	1.72	1.72	1.65	1.33	1.19	1.03	1.01	1.01	1.00

Table 8 Average optimality gap for instances with $m = 2$, $d = 5$

Instance type				AP formulations	Average optimality gap at time t (s)								
m	d	$Q_{\max}$ range	n		5	10	30	60	150	300	600	1200	1800
2	5	low	8	basic	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
				enhanced	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
			12	basic	1.51	1.40	1.35	1.26	1.08	1.00	1.00	1.00	1.00
				enhanced	1.58	1.41	1.30	1.19	1.02	1.01	1.00	1.00	1.00
			16	basic	1.74	1.70	1.59	1.56	1.51	1.45	1.40	1.36	1.34
				enhanced	1.77	1.73	1.57	1.53	1.47	1.41	1.36	1.31	1.29
			20	basic	1.77	1.76	1.73	1.69	1.66	1.62	1.58	1.55	1.53
				enhanced	1.78	1.78	1.71	1.66	1.63	1.59	1.55	1.51	1.50
		intermediate	8	basic	1.38	1.36	1.24	1.20	1.15	1.11	1.07	1.05	1.04
				enhanced	1.39	1.35	1.20	1.14	1.08	1.06	1.03	1.00	1.00
			12	basic	1.80	1.64	1.48	1.44	1.36	1.33	1.30	1.27	1.25
				enhanced	1.81	1.78	1.46	1.41	1.31	1.27	1.25	1.22	1.20
			16	basic	1.81	1.81	1.65	1.48	1.46	1.39	1.36	1.34	1.33
				enhanced	1.81	1.81	1.80	1.62	1.46	1.36	1.32	1.29	1.28
			20	basic	NA	1.84	1.84	1.82	1.54	1.53	1.49	1.43	1.42
				enhanced	NA	1.85	1.84	1.83	1.71	1.51	1.45	1.40	1.39
		high	8	basic	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
				enhanced	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
			12	basic	1.37	1.07	1.01	1.00	1.00	1.00	1.00	1.00	1.00
				enhanced	1.81	1.47	1.03	1.00	1.00	1.00	1.00	1.00	1.00
			16	basic	1.77	1.36	1.03	1.01	1.00	1.00	1.00	1.00	1.00
				enhanced	1.81	1.81	1.60	1.08	1.00	1.00	1.00	1.00	1.00
			20	basic	NA	1.84	1.54	1.17	1.03	1.01	1.00	1.00	1.00
				enhanced	NA	1.84	1.84	1.80	1.10	1.03	1.00	1.00	1.00

NA: for these instances, the root LP relaxations of the formulations did not solve in less than 5 s

Table 9 Average optimality gap for instances with $m = 4$, $d = 2$

Instance type				AP formulations	Average optimality gap at time t (s)								
m	d	Q_{max} range	n		5	10	30	60	150	300	600	1200	1800
4	2	low	8	basic	2.53	2.43	2.27	2.23	2.15	2.10	2.03	1.99	1.96
				enhanced	2.50	2.40	2.26	2.22	2.12	2.07	2.03	1.96	1.94
			12	basic	2.78	2.75	2.72	2.64	2.56	2.52	2.47	2.43	2.41
				enhanced	NA	2.78	2.66	2.64	2.58	2.53	2.47	2.43	2.41
			16	basic	NA	3.02	2.93	2.92	2.88	2.83	2.81	2.75	2.73
				enhanced	NA	3.02	2.96	2.89	2.87	2.83	2.79	2.73	2.71
			20	basic	NA	3.08	3.04	3.03	3.00	2.98	2.95	2.92	2.91
				enhanced	NA	3.08	3.05	3.03	2.98	2.96	2.94	2.91	2.89
		intermediate	8	basic	2.76	2.73	2.11	2.06	1.86	1.65	1.57	1.54	1.51
				enhanced	2.76	2.76	1.89	1.86	1.64	1.56	1.51	1.48	1.46
			12	basic	NA	3.01	3.01	3.00	2.52	2.51	2.33	2.04	1.98
				enhanced	NA	3.01	3.00	2.93	2.34	2.32	2.01	1.89	1.84
			16	basic	NA	3.09	3.09	3.09	3.09	3.00	2.74	2.73	2.70
				enhanced	NA	3.09	3.09	3.09	3.08	2.67	2.58	2.40	2.20
			20	basic	NA	3.17	3.17	3.17	3.17	3.17	3.17	2.98	2.88
				enhanced	NA	3.17	3.17	3.17	3.17	3.17	3.16	2.99	2.88
		high	8	basic	2.76	2.67	1.79	1.74	1.48	1.12	1.01	1.00	1.00
				enhanced	2.76	2.75	1.61	1.49	1.19	1.06	1.02	1.00	1.00
			12	basic	NA	3.01	3.01	2.94	2.18	2.18	1.59	1.24	1.20
				enhanced	NA	3.01	3.00	2.97	1.94	1.71	1.17	1.08	1.06
			16	basic	NA	3.09	3.09	3.09	3.09	2.91	2.64	2.63	2.48
				enhanced	NA	3.09	3.09	3.09	3.09	2.31	1.78	1.40	1.25
			20	basic	NA	3.17	3.17	3.17	3.17	3.17	3.17	3.00	2.81
				enhanced	NA	3.17	3.17	3.17	3.17	3.17	3.13	2.67	2.19

NA: for these instances, the root LP relaxations of the formulations did not solve in less than 5 s

(b) *There exists an optimal schedule for Problem 2.3 such that for each block (j,k) of jobs, the total processing time $p(j,k) = \min\{p_1(j,k), p_2(j,k)\}$.*

Note that $p(j,k)$ is not necessarily equal to $p(k,j)$. Using the above lemma, we obtain the following result.

Theorem 4.3 *Problem 2.3 can be viewed as an instance of the asymmetric traveling salesperson problem.*

Proof For convenience, we introduce a new job 0 with $p_{10} = p_{20} = 0$, and define $p(0,j) = p_{1js^*_{1j}}$ and $p(j,0) = p_{2js^*_{2j}}$ for $j = 1,\ldots,n$. Denote $j_0 = j_{n+1} = 0$. Then the makespan of schedule $(j_1, j_2, \ldots, j_n)$ is equal to $\sum_{i=0}^{n} p(j_i, j_{i+1})$. We construct a complete graph $G = (V, E)$ in which $V = \{0, 1, \ldots, n\}$. We define the distance from node j to k as $D_{jk} = p(j,k)$ for $j,k \in \{0,\ldots,n\}$ such that $j \neq k$, and $D_{jj} = +\infty$ for $j = 0,\ldots,n$. Then Problem 2.3 is equivalent to finding the shortest tour in G with arc distances D. □

5 Two machines, continuous speeds, and zero intermediate storage

In this section we will consider Problem 2.4, in which the flow shop has two machines with zero intermediate storage, each machine can process jobs at any speed within a continuous interval, and the power consumption of a machine processing a job at speed s is s^α for some constant $\alpha > 1$. Given $Q_{\max}$ as the threshold for peak power consumption, we define $s_{\max} = (Q_{\max})^{\frac{1}{\alpha}}$, and let the speed set $\mathcal{S}$ be the continuous interval $[0, s_{\max}]$. Recall that p_{ij} is the work required for job j on machine i, $s_{ij} \in [0, s_{\max}]$ is the chosen speed to process job j on machine i, and p_{ij}/s_{ij} is the processing time of job j on machine i.

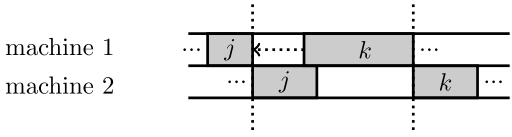
5.1 Arbitrary work requirements across machines

Lemma 5.1 *In any optimal schedule for Problem 2.4, if job j immediately precedes job k, then $s^\alpha_{1k} + s^\alpha_{2j} = Q_{\max} = s^\alpha_{\max}$. Moreover, each block (j,k) with $j \neq 0$ and $k \neq 0$ in an optimal schedule is processed with overlap, and $C_{2j} = C_{1k}$.*

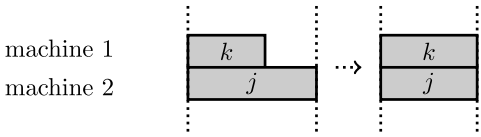
Proof Note that at any time, the total power consumption of the two machines must be exactly $Q_{\max}$; otherwise we can increase the speeds of the jobs on the two machines so that the total power consumption is $Q_{\max}$, and decrease the makespan.

Consider a block (j,k) with $j \neq 0$ and $k \neq 0$ in an optimal schedule. If block (j,k) is processed without overlap in the optimal schedule, then job j and k must be processed at the maximum speed $s_{\max}$. That is, the minimum total processing time for block (j,k) without overlap is $p_1(j,k) = \frac{p_{2j}+p_{1k}}{(Q_{\max})^{\frac{1}{\alpha}}}$.

If block (j,k) is processed with overlap in the optimal schedule, then it must be nondelay; otherwise we can process the delayed job (job k) earlier with the same speed as follows:



This way, we can decrease the makespan without violating the power consumption constraints. As a result, when block (j,k) is processed with overlap, then job j on machine 2 and job k on machine 1 must be processed at the same start time. Moreover, we also have $C_{2j} = C_{1k}$. Otherwise, without loss of generality, suppose that $C_{2j} > C_{1k}$. Then we can decrease the speed of job k and increase the speed of job j until $C_{2j} = C_{1k}$.



This way, the total processing time of block (j,k) decreases, and so the makespan also decreases. Therefore for any block (j,k) with overlap, we have $\frac{p_{2j}}{s_{2j}} = \frac{p_{1k}}{s_{1k}} = p_2(j,k)$. Because $s_{2j}^{\alpha} + s_{1k}^{\alpha} = Q_{\max}$, we obtain that $p_2(j,k) = \frac{(p_{1k}^{\alpha}+p_{2j}^{\alpha})^{\frac{1}{\alpha}}}{(Q_{\max})^{\frac{1}{\alpha}}}$. Since $p_1(j,k) > p_2(j,k)$ for any $\alpha > 1$, so we have $p(j,k) = p_2(j,k)$. □

Define $D_{jk} = (p_{1k}^{\alpha} + p_{2j}^{\alpha})^{\frac{1}{\alpha}}$ for $j,k \in \{0,\ldots,n\}$ such that $j \neq k$. Since $p(j,k) = \frac{D_{jk}}{(Q_{\max})^{\frac{1}{\alpha}}}$ for all $j,k \in \{0,\ldots,n\}$ such that $j \neq k$, our problem is equivalent to finding a permutation $(j_1,\ldots,j_n)$ of the jobs that minimizes $\sum_{i=0}^{n} D_{j_i,j_{i+1}}$. Similar to the proof in Theorem 4.3, if we interpret D_{jk} as the distance of the arc from node j to node k in a complete directed graph on $\{0,1,\ldots,n\}$, and define $D_{jj} = +\infty$ for $j = 0,\ldots,n$, then this variant of the PFSPP problem is also a special case of the asymmetric TSP.

5.2 Consistent work requirements across machines

Although the asymmetric TSP is an NP-hard problem, many of its special cases can be solved efficiently in polynomial time. One such special case is when the arc distances satisfy the so-called *Demidenko conditions*, which state that the matrix $D \in \mathbb{R}^{(n+1)\times(n+1)}$ of arc distances satisfies the following conditions: for all $i,j,l \in \{0,1,\ldots,n\}$ such that $i < j < j+1 < l$, we have

$$D_{ij} + D_{j,j+1} + D_{j+1,l} \leq D_{i,j+1} + D_{j+1,j} + D_{jl}, \tag{5.1}$$

$$D_{l,j+1} + D_{j+1,j} + D_{ji} \leq D_{lj} + D_{j,j+1} + D_{j+1,i}, \tag{5.2}$$

$$D_{ij} + D_{l,j+1} \leq D_{lj} + D_{i,j+1}, \tag{5.3}$$

$$D_{ji} + D_{j+1,l} \leq D_{jl} + D_{j+1,i}. \tag{5.4}$$

We say that a tour on cities $0,1,\ldots,n$ is *pyramidal* if it is of the form $(0,i_1,\ldots,i_r,n,j_1,\ldots,j_{n-r-1})$, where $i_1 < i_2 < \cdots < i_r$ and $j_1 > j_2 > \cdots > j_{n-r-1}$. Demidenko () showed the following for the asymmetric TSP.

Theorem 5.2 (Demidenko) *If $D \in \mathbb{R}^{(n+1)\times(n+1)}$ satisfies the Demidenko conditions, then for any tour there exists a pyramidal tour of no greater cost. Moreover, a minimum cost pyramidal tour can be determined in $O(n^2)$ time.*

Coming back to Problem 2.4, suppose the work requirement of jobs is *consistent* across machines: that is, for any two jobs $j, k \in \mathcal{J}$, we have that $p_{1j} \le p_{1k}$ implies $p_{2j} \le p_{2k}$. Then we have the following theorem.

Theorem 5.3 *If the work required is consistent across machines, then there exists an optimal schedule for Problem 2.4 that corresponds to a pyramidal TSP tour, and such a schedule can be found in $O(n^2)$ time.*

Proof Fix $i, j, l \in \{0, 1, \ldots, n\}$ such that $i < j < j+1 < l$. Without loss of generality, suppose $p_{11} \le p_{12} \le \cdots \le p_{1n}$ and $p_{21} \le p_{22} \le \cdots \le p_{2n}$. We can do this since the work is assumed to be consistent across machines. Therefore, $p_{1i} \le p_{1j} \le p_{1,j+1} \le p_{1l}$ and $p_{2i} \le p_{2j} \le p_{2,j+1} \le p_{2l}$. We prove that $D_{jk} = (p_{1k}^{\alpha} + p_{ij}^{\alpha})^{\frac{1}{\alpha}}$ for $j, k \in \{0, 1, \ldots, n\}$ such that $j \ne k$ and $D_{jj} = +\infty$ for $j = 0, 1, \ldots, n$ satisfies the Demidenko conditions.

Conditions (5.1): Let $g(x) = (x^{\alpha} + p_{2i}^{\alpha})^{\frac{1}{\alpha}} - (x^{\alpha} + p_{2,j+1}^{\alpha})^{\frac{1}{\alpha}}$. Then it is straightforward to verify that $g'(x) \ge 0$, and so we have $g(p_{1,j+1}) \ge g(p_{1j})$, i.e.

$$\left(p_{1,j+1}^{\alpha} + p_{2i}^{\alpha}\right)^{\frac{1}{\alpha}} - \left(p_{1,j+1}^{\alpha} + p_{2,j+1}^{\alpha}\right)^{\frac{1}{\alpha}} \ge \left(p_{1j}^{\alpha} + p_{2i}^{\alpha}\right)^{\frac{1}{\alpha}} - \left(p_{1j}^{\alpha} + p_{2,j+1}^{\alpha}\right)^{\frac{1}{\alpha}}.$$

This is equivalent to

$$\begin{aligned}&\left(p_{1,j+1}^{\alpha} + p_{2i}^{\alpha}\right)^{\frac{1}{\alpha}} - \left(p_{1j}^{\alpha} + p_{2i}^{\alpha}\right)^{\frac{1}{\alpha}} + \left(p_{1j}^{\alpha} + p_{2,j+1}^{\alpha}\right)^{\frac{1}{\alpha}} - \left(p_{1,j+1}^{\alpha} + p_{2j}^{\alpha}\right)^{\frac{1}{\alpha}}\\ &\quad\ge \left(p_{1,j+1}^{\alpha} + p_{2,j+1}^{\alpha}\right)^{\frac{1}{\alpha}} - \left(p_{1,j+1}^{\alpha} + p_{2j}^{\alpha}\right)^{\frac{1}{\alpha}}.\end{aligned}$$

Let $f(x) = (x^{\alpha} + p_{2,j+1}^{\alpha})^{\frac{1}{\alpha}} - (x^{\alpha} + p_{2j}^{\alpha})^{\frac{1}{\alpha}}$. Similarly we can prove that $f'(x) \le 0$, and so

$$\left(p_{1,j+1}^{\alpha} + p_{2,j+1}^{\alpha}\right)^{\frac{1}{\alpha}} - \left(p_{1,j+1}^{\alpha} + p_{2j}^{\alpha}\right)^{\frac{1}{\alpha}} \ge \left(p_{1l}^{\alpha} + p_{2,j+1}^{\alpha}\right)^{\frac{1}{\alpha}} - \left(p_{1l}^{\alpha} + p_{2j}^{\alpha}\right)^{\frac{1}{\alpha}}.$$

So we have

$$\begin{aligned}&\left(p_{1,j+1}^{\alpha} + p_{2i}^{\alpha}\right)^{\frac{1}{\alpha}} - \left(p_{1j}^{\alpha} + p_{2i}^{\alpha}\right)^{\frac{1}{\alpha}} + \left(p_{1j}^{\alpha} + p_{2,j+1}^{\alpha}\right)^{\frac{1}{\alpha}} - \left(p_{1,j+1}^{\alpha} + p_{2j}^{\alpha}\right)^{\frac{1}{\alpha}}\\ &\quad\ge \left(p_{1l}^{\alpha} + p_{2,j+1}^{\alpha}\right)^{\frac{1}{\alpha}} - \left(p_{1l}^{\alpha} + p_{2j}^{\alpha}\right)^{\frac{1}{\alpha}},\end{aligned}$$

or equivalently

$$D_{i,j+1} - D_{ij} + D_{j+1,j} - D_{j,j+1} \ge D_{j+1,l} - D_{jl},$$

which indicates that conditions (5.1) are satisfied.

Conditions (5.2): Similar to the proof of conditions (5.1).

Conditions (5.3): Let $h(x) = (x^{\alpha} + p_{2i}^{\alpha})^{\frac{1}{\alpha}} - (x^{\alpha} + p_{2l}^{\alpha})^{\frac{1}{\alpha}}$. Then it is straightforward to verify that $h'(x) \ge 0$, so we have

$$\left(p_{1,j+1}^{\alpha} + p_{2i}^{\alpha}\right)^{\frac{1}{\alpha}} - \left(p_{1,j+1}^{\alpha} + p_{2l}^{\alpha}\right)^{\frac{1}{\alpha}} \ge \left(p_{1j}^{\alpha} + p_{2i}^{\alpha}\right)^{\frac{1}{\alpha}} - \left(p_{1j}^{\alpha} + p_{2l}^{\alpha}\right)^{\frac{1}{\alpha}},$$

which indicates that conditions (5.3) are satisfied.

Conditions (5.4): Similar to the proof of conditions (5.3). □

In general, it may be the case that different jobs have their own speed ranges and power functions (e.g. Bansal et al. 2009). In other words, it may be the case that each job j has a power function of the form $a_j s_j^{\alpha} + c_j$, where $s_j \in [s_j^{\min}, s_j^{\max}] \triangleq \mathcal{S}_j$. Under this environment, we may obtain different functions $p(j,k)$ with respect to p_{1k} and p_{2j} for each block (j,k) in Problem 2.4. Using the Demidenko conditions, we can extend Theorem 5.3 as follows.

Theorem 5.4 *For Problem 2.4, if the functions $p(j,k)$ for all $j,k \in \{0,1,\ldots,n\}$ with $j \neq k$ are determined by a twice differentiable function $g(x,y)$ such that $p(j,k) = g(p_{1k}, p_{2j})$ and $\frac{\partial^2 g}{\partial x \partial y} < 0$, then there must exist an optimal schedule that corresponds to a pyramidal tour.*

Proof Fix $i,j,l \in \{0,1,\ldots,n\}$ such that $i < j < j+1 < l$. Without loss of generality, suppose $p_{11} \leq p_{12} \leq \cdots \leq p_{1n}$ and $p_{21} \leq p_{22} \leq \cdots \leq p_{2n}$. Similar to the proof of Theorem 5.3, we show that the matrix D such that $D_{jk} = p(j,k)$ for all $j,k \in \{0,1,\ldots,n\}$ such that $j \neq k$ satisfies the Demidenko conditions under the above assumption.

To show conditions (5.1) are satisfied, we need to prove that

$$\begin{aligned} &g(p_{1,j+1}, p_{2i}) + g(p_{1j}, p_{2,j+1}) + g(p_{1l}, p_{2j}) \\ &\quad \geq g(p_{1j}, p_{2i}) + g(p_{1,j+1}, p_{2j}) + g(p_{1l}, p_{2,j+1}). \end{aligned}$$

Let $h(x) = g(x, p_{2i}) - g(x, p_{2,j+1})$. Then $\frac{\partial h}{\partial x} = \frac{\partial g(x,p_{2i})}{\partial x} - \frac{\partial g(x,p_{2,j+1})}{\partial x}$. Because $\frac{\partial^2 g}{\partial x \partial y} < 0$, we obtain that $\frac{\partial h}{\partial x} \geq 0$. So

$$g(p_{1,j+1}, p_{2i}) - g(p_{1,j+1}, p_{2,j+1}) \geq g(p_{1j}, p_{2i}) - g(p_{1j}, p_{2,j+1}).$$

Similarly, we can also prove that

$$g(p_{1,j+1}, p_{2,j+1}) - g(p_{1,j+1}, p_{2j}) \geq g(p_{1l}, p_{2,j+1}) - g(p_{1l}, p_{2j}).$$

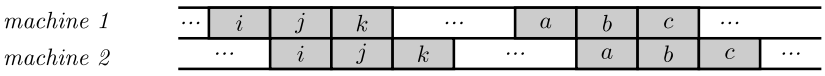
Combining the above two results, conditions (5.1) are satisfied.

Using the same arguments as in the proof of Theorem 5.3 and above, we can verify conditions (5.2), (5.3), and (5.4) similarly. □

5.3 Equal work requirements across machines

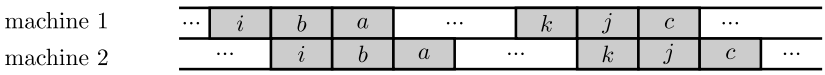
If the work required for each job is equal on each machine—that is, for any job $j \in \mathcal{J}$, we have that $p_{1j} = p_{2j} = p_j$—then we can further refine the results of the previous subsection. By Theorem 5.3, there exists an optimal pyramidal tour for this variant of Problem 2.4. For this variant, we claim that there must exist an optimal schedule of the form $(1,3,5,\ldots,n,\ldots,6,4,2)$, assuming that $p_1 \leq p_2 \leq \cdots \leq p_n$.

Lemma 5.5 *Consider a subsequence of an optimal schedule as follows*:



If $p_i \leq p_c$, then we must have $p_j \leq p_b$.

Proof By contradiction. Suppose in an optimal schedule σ_1, we have $p_i \leq p_c$ but $p_j > p_b$. Consider rescheduling the jobs between job i and c in reverse order as follows (i.e. $i \to b \to a \to \cdots \to k \to j \to c$). Denote this new schedule as σ_2.

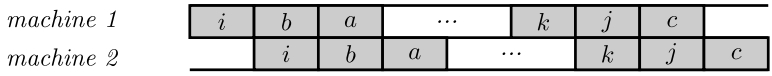


Denote the makespan of schedule σ_i as C_{σ_i}, $i = 1, 2$. Then we have

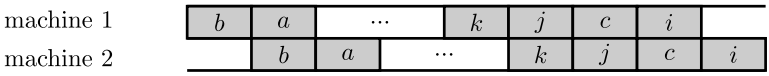
$$C_{\sigma_1} - C_{\sigma_2} = \frac{(p_i^\alpha + p_j^\alpha)^{\frac{1}{\alpha}} + (p_c^\alpha + p_b^\alpha)^{\frac{1}{\alpha}} - (p_i^\alpha + p_b^\alpha)^{\frac{1}{\alpha}} - (p_c^\alpha + p_j^\alpha)^{\frac{1}{\alpha}}}{(Q_{\max})^{\frac{1}{\alpha}}}.$$

Similar to the proof of Theorem 5.3, we can show that $C_{\sigma_1} - C_{\sigma_2} > 0$, which contradicts schedule σ_1 being optimal. □

Lemma 5.6 *For any optimal schedule, suppose that the first job processed is job i, the second is b and the last is c. Without loss of generality, if we assume that $p_i \le p_c$, then there must exist an optimal schedule which satisfies $p_b \ge p_c$.*



Proof By contradiction. If an optimal schedule σ_1 does not satisfy $p_b \ge p_c$, i.e. $p_b < p_c$, then we reschedule job i so that i becomes the last job in the schedule, while maintaining the ordering of all the other jobs. We denote the new schedule as σ_2:



Similar to the proof of Theorem 5.3, it is easy to verify that

$$C_{\sigma_1} - C_{\sigma_2} = \frac{p_c + (p_i^\alpha + p_b^\alpha)^{\frac{1}{\alpha}} - p_b - (p_i^\alpha + p_c^\alpha)^{\frac{1}{\alpha}}}{(Q_{\max})^{\frac{1}{\alpha}}} > 0,$$

and so schedule σ_1 is not optimal. □

Theorem 5.7 *Assuming that $p_1 \le p_2 \le \cdots \le p_n$, there must exist an optimal schedule of the form* $(1, 3, 5, \ldots, n, \ldots, 6, 4, 2)$.

Proof For simplicity, we denote the workload of the job in jth position of an optimal schedule as $p_{\sigma(j)}$. Without loss of generality, we assume that $p_{\sigma(1)} \le p_{\sigma(n)}$. By Lemma 5.5, we obtain that $p_{\sigma(i)} \le p_{\sigma(n-i+1)}$ for $i = 1, \ldots, \lfloor \frac{n}{2} \rfloor$. By Lemma 5.6, we have $p_{\sigma(2)} \ge p_{\sigma(n)}$. Consider the subsequence of an optimal schedule from the second position to the nth position. By Lemma 5.5, we obtain that $p_{\sigma(i+1)} \ge p_{\sigma(n-i+1)}$ for $i = 1, \ldots, \lfloor \frac{n}{2} \rfloor$. Combining the above results, we have $p_{\sigma(1)} \le p_{\sigma(n)} \le p_{\sigma(2)} \le p_{\sigma(n-1)} \le p_{\sigma(3)} \le \cdots$, and so there exists an optimal schedule of the form $(1, 3, 5, \ldots, n, \ldots, 6, 4, 2)$. □

6 Conclusions and future work

To the best of our knowledge, our paper is one of the first to consider a multi-objective flow shop scheduling problem with traditional time-based objectives (i.e. makespan) as well as energy-based objectives (i.e. peak power consumption). In particular, in this paper, we

studied the permutation flow shop problem with peak power consumption constraints (the PFSPP problem). We proposed two mixed integer programming formulations and accompanying valid inequalities for the case of discrete speeds. A key feature of our formulation is variables and constraints that keep track of jobs running concurrently across machines. This may be of interest in other applications.

We investigated the computational performance of these formulations with instances arising from the manufacture of cast iron plates. Although our valid inequalities for the assignment and positional formulation resulted in better computational performance, especially for small-to-moderate sized instances, we still had difficulty obtaining optimal schedules in a reasonable amount of time for instances with large numbers of jobs and machines. One potential direction for future research is to develop stronger valid inequalities for our formulations, in the hopes of strengthening these formulations and improving their computational performance.

We also showed that our scheduling problem can be recast as an asymmetric TSP when the flow shop has two machines with zero intermediate storage. In addition, we were able to obtain stronger structural characterizations of optimal schedules and polynomial time algorithms to find these schedules when the speed set is continuous and the work requirements satisfy certain conditions.

Of course, there are many other possible directions for future research stemming from this work. For example, the computational complexity of the PFSPP problem remains open when there are two machines. It would be interesting to fully characterize which two-machine variants of our scheduling problem are NP-hard or polynomial time solvable. Since minimizing the makespan in an ordinary permutation flow shop is NP-hard when there are three or more machines (Garey et al. 1976) and the PFSPP problem can be seen as a special case of this ordinary permutation flow shop problem (by setting the peak power consumption threshold $Q_{\max}$ sufficiently high), the PFSPP problem is also NP-hard when there are three or more machines. Another interesting direction would be to determine when our proposed formulations have strong LP relaxations. Last but not least, it would also be interesting to consider different time or energy objectives (e.g. total weighted completion time, carbon footprint) or some other complex machine environments with peak power consumption constraints.

References

Albers, S. (2010). Energy-efficient algorithms. *Communications of the ACM*, *53*(5), 86–96.

Babu, C. A., & Ashok, S. (2008). Peak load management in electrolytic process industries. *IEEE Transactions on Power Systems*, *23*(2), 399–405.

Bansal, N., Kimbrel, T., & Pruhs, K. (2007). Speed scaling to manage energy and temperature. *Journal of the ACM*, *54*(1), 1–39.

Bansal, N., Chan, H. L., & Pruhs, K. (2009). Speed scaling with an arbitrary power function. In *Proceedings of the 20th annual ACM-SIAM symposium on discrete algorithms* (pp. 693–701).

Bouzid, W. (2005). Cutting parameter optimization to minimize production time in high speed turning. *Journal of Materials Processing Technology*, *161*(3), 388–395.

Cochran, R., Hankendi, C., Coskun, A., & Reda, S. (2011). Pack & cap: adaptive DVFS and thread packing under power caps. In *Proceedings of the 44th annual IEEE/ACM international symposium on microarchitecture*.

Dahmus, J. B., & Gutowski, T. G. (2004). An environmental analysis of machining. In *ASME 2004 international mechanical engineering congress and exposition* (pp. 643–652).

Demidenko, V. M. (1979). The traveling salesman problem with asymmetric matrices. *Izvestiâ Akademii Nauk BSSR. Seriâ Fiziko-Matematičeskih Nauk*, *1*, 29–35 (in Russian).

Drake, R., Yildirim, M. B., Twomey, J., Whitman, L., Ahmad, J., & Lodhia, P. (2006). Data collection framework on energy consumption in manufacturing. In *IIE annual conference and expo 2006*.

Fang, K., Uhan, N. A., Zhao, F., & Sutherland, J. W. (2011). A new approach to scheduling in manufacturing for power consumption and carbon footprint reduction. *Journal of Manufacturing Systems*, *30*(4), 234–240.

Felter, W., Rajamani, K., Keller, T., & Rusu, C. (2005). A performance-conserving approach for reducing peak power consumption in server systems. In *Proceedings of the 19th annual international conference on supercomputing* (pp. 293–302).

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, *1*(2), 117–129.

Gutowski, T., Murphy, C., Allen, D., Bauer, D., Bras, B., Piwonka, T., Sheng, P., Sutherland, J., Thurston, D., & Wolff, E. (2005). Environmentally benign manufacturing: observations from Japan, Europe and the United States. *Journal of Cleaner Production*, *13*, 1–17.

Irani, S., & Pruhs, K. R. (2005). Algorithmic problems in power management. *SIGACT News*, *36*(2), 63–76.

Keha, A. B., Khowala, K., & Fowler, J. W. (2009). Mixed integer programming formulations for single machine scheduling problems. *Computers & Industrial Engineering*, *56*(1), 357–367.

Kontorinis, V., Shayan, A., Tullsen, D. M., & Kumar, R. (2009). Reducing peak power with a table-driven adaptive processor core. In *Proceedings of the 42nd annual IEEE/ACM international symposium on microarchitecture* (pp. 189–200).

Kwon, W. C., & Kim, T. (2005). Optimal voltage allocation techniques for dynamically variable voltage processors. *ACM Transactions on Embedded Computing Systems*, *4*(1), 211–230.

Lasserre, J. B., & Queyranne, M. (1992). Generic scheduling polyhedra and a new mixed-integer formulation for single-machine scheduling. In *Proceedings of the 2nd integer programming and combinatorial optimization conference* (pp. 136–149).

Manne, A. S. (1960). On the job-shop scheduling problem. *Operations Research*, *8*(2), 219–223.

Mouzon, G., & Yildirim, M. B. (2008). A framework to minimise total energy consumption and total tardiness on a single machine. *International Journal of Sustainable Engineering*, *1*(2), 105–116.

Mouzon, G., Yildirim, M. B., & Twomey, J. (2007). Operational methods for minimization of energy consumption of manufacturing equipment. *International Journal of Production Research*, *45*(18–19), 4247–4271.

Mudge, T. (2001). Power: a first-class architectural design constraint. *Computer*, *34*(4), 52–58.

Oberg, E., Jones, F. D., Horton, H. L., & Ryffel, H. H. (2008). *Machinery's handbook* (28th ed.). New York: Industrial Press.

Reddi, S. S., & Ramamoorthy, C. V. (1972). On the flow-shop sequencing problem with no wait in process. *Operational Research Quarterly*, *23*(3), 323–331.

Stafford, E. F. Jr., Tseng, F. T., & Gupta, J. N. D. (2005). Comparative evaluation of MILP flowshop models. *Journal of the Operational Research Society*, *56*, 88–101.

Thörnblad, K., Strömberg, A. B., & Patriksson, M. (2010). Optimization of schedules for a multitask production cell. In *The 22nd annual NOFOMA conference proceedings*.

Unlu, Y., & Mason, S. J. (2010). Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. *Computers & Industrial Engineering*, *58*(4), 785–800.

Wagner, H. M. (1959). An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, *6*(2), 134–140.

Yao, F., Demers, A., & Shenker, S. (1995). A scheduling model for reduced CPU energy. In *Proceedings of the 36th annual symposium on foundations of computer science* (pp. 374–382).