Propagation and branching strategies for job shop scheduling minimizing the weighted energy consumption

Andreas Bley and Andreas Linß

Universität Kassel, Institut für Mathematik, Heinrich-Plett-Straße 40, 34132 Kassel, Germany, andreas.bley@uni-kassel.de, andreas.linss@uni-kassel.de

Abstract. We consider a job shop scheduling problem with time windows, flexible energy prices, and machines whose energy consumption depends on their operational state (offline, ramp-up, setup, processing, standby or ramp-down). The goal is to find a valid schedule that minimizes the overall energy cost. To solve this problem to optimality, we developed a branch-and-bound algorithm based on a time-indexed integer linear programming (ILP) formulation, which uses binary variables that describe blocks spanning multiple inactive periods on the machines. In this paper, we discuss the propagation and branching schemes used in that algorithm. The strategies, which are specifically tailored for energy related machine scheduling problems, primarily aim to determine and sharpen the activity profiles of the machines (and thus reduce the number of the inactive block variables) and address the workload profile of the tasks with lower priority. Computational experiments validate the efficiency of those techniques.

Keywords: integer programming, machine scheduling, presolving, branch and bound

1 Introduction

In recent years, energy awareness and increasing energy prices gained a lot of attention in production planning. Various approaches to incorporate energy consumption or costs into models and solution techniques for machine scheduling problems have been proposed, see for example [7, 11]. Those models explicitly consider different machine states, such as *processing*, *standby* or *off*, the transitions among these states, and the respective energy demands and durations [3, 13]. Problems with time-dependent energy costs are typically modeled using time-indexed formulations, which leads to huge ILP models. Even more, in contrast to classical objectives minimizing completion times, minimizing the energy consumption leads to highly fractional solutions of the linear programming (LP) relaxations. Thus, tailored model reduction and branching techniques are needed to solve those models efficiently.

In this paper, we present such techniques for a variant of the job-shop scheduling problem with flexible energy prices and time windows discussed in [5, 12]. An overview of formulations for the classical job-shop scheduling problem is given in [9]. A survey on alternative modelling and solution approaches for job-shop scheduling can be found in [14].

2 Problem description and formulation

In our problem, the planning horizon $[T] := \{0, \dots, T-1\}$ consists of T uniform periods. For each $t \in [T]$, we are given an energy price $C_t \in \mathbb{R}_{\geq 0}$, which is valid during period t. Furthermore, we are given a set of (non-uniform) machines $M := \{1, \dots, n_M\}$ and a set of jobs $J := \{1, \dots, n_J\}$. A job $j \in J$ consists of a list of tasks (j,k), $j \in J$, $k \in O_j := [n_j]$, $n_j \in \mathbb{N}$, which must be processed in the order defined by O_j. Each task (j,k) must be setup and processed on a predefined machine $m_{j,k} \in M$. For each task (j,k), we are given its setup time $d^{se}_{j,k} \in \mathbb{N}$ and its processing time $d^{pr}_{j,k} \in \mathbb{N}$. In addition, we are given a release date $a_j \in [T]$ and a due date $f_j \in [T]$ for each job $j \in J$, which apply to the first and the last task of the job, respectively. We let $O := \{(j,k) : j \in J,\, k \in O_j\}$. $O^M_{|_m} = \{(j,k) \in O \mid m_{j,k} = m\}$ denotes the set of tasks (j,k) on machine $m \in M$.

In each period $t \in [T]$, each machine $m \in M$ must be in one of the operating states *off*, *processing*, *setup*, *standby*, *ramp-up* or *ramp-down*, summarized as $\mathcal{S} = \{$*off*, *pr*, *se*, *st*, *ru*, *rd*$\}$. A machine is called active if its operating state is *setup*, *processing*, or *standby*, otherwise it is called inactive, with the canonical switches between the states and implications between tasks and machine states. The duration of the *ramp-up* phase, changing from *off* to any state $s \in \{$*se*, *pr*, *st*, *rd*$\}$, is $d^{ru}_m \in \mathbb{N}$. The duration of the *ramp-down* phase is $d^{rd}_m \in \mathbb{N}$. For each machine $m \in M$ and state $s \in \mathcal{S}$, D^s_m is the energy demand of machine m in state s.

A feasible solution consists of the start time for each task's processing and a machine state for each machine and each period. Each task is processed non-preemptively and each task's setup immediately precedes (also non-preemptively) its processing. The processing of a task can start only after the processing of its predecessor has been completed, but its setup can already start while the predecessor is processing (on another machine). The start of the first and the completion of the last task of each job must obey this job's release and due dates, respectively. Only one task can be processed or set up on a machine simultaneously. A machine processing or setting up for a task must be in state *processing* or *setup*, respectively. Otherwise, the machine can be active in *standby* or become inactive *ramping down*, being *off*, or *ramping up*, respecting the ramping durations and canonical state relations. At the beginning and the end of the planning horizon, each machine must be *off*. Our goal is to find a solution whose energy cost is minimized.

From the task precedences, the ramping, setup and processing times, and the jobs' release and due dates, we obtain for each task $(j,k) \in O^M_{|_m}$ on machine m the earliest period $a_{j,k} := \max\{a_j, d^{ru}_m + d^{se}_{j,0}\} + \sum_{q=0}^{k-1} d^{pr}_{j,q}$ and the latest period $f_{j,k} := \min\{f_j, T - d^{rd}_m\} - \sum_{l=k}^{|O_j|-1} d^{pr}_{j,l}$ when its processing may start.

We use a time-indexed formulation with binary variables to explicitly indicate so-called breaks, i.e., inactive blocks of consecutive *ramp-down*–*off*–*ramp-up* periods on the machines, to avoid inequalities describing the ramping mechanism. To use this type of variables also to model the initial and final ramping, we extend the time window for each machine to $T_+^m := \{-d_m^{rd}, \dots, T + d_m^{ru} - 1\}$ and enforce that the machine is *off* in periods 0 and T. The energy price is set to $C_t = 0$ for the artificial periods $t \in T_+^m \setminus T$. For each machine m, B_m denotes the set of all feasible breaks (t_0, t_1), $t_0, t_1 \in [T_+^m]$ and $t_1 - t_0 \geq d_m^{ru} + d_m^{rd}$, where the machine is starting its *ramp-down* at t_0, is *off* from $t_0 + d_m^{rd}$ until $t_1 - d_m^{ru} - 1$, and in *ramp-up* from $t_1 - d_m^{ru}$ to $t_1 - 1$. In total, we have three types of variables:

- $x_{j,k,t} \in \{0,1\}$ indicating iff task $(j,k) \in O$ starts processing in period $t \in [T]$,
- $z_{m,t}^{st} \in \{0,1\}$ indicating iff machine $m \in M$ is in state *standby* in $t \in [T]$,
- $z_{t_0,t_1}^{m,br} \in \{0,1\}$ indicating iff $m \in M$ is in a break from t_0 to t_1, $(t_0,t_1) \in B_m$.

The associated objective coefficients $\hat{c}_{j,k,t}$ and $\hat{c}_{m,t_0,t_1}^{br}$ express the total energy cost induced by the setup and processing of task (j,k) if started in period t and by a break from t_0 until t_1 on machine m, respectively.

We obtain the following integer programming formulation of the problem:

$$\min \sum_{m \in M} \Big(\sum_{t \in [T]} \big(C_t D_m^{st} z_{m,t}^{st} + \sum_{(j,k) \in O_{|m}^M} \hat{c}_{j,k,t} x_{j,k,t}\big) + \sum_{(t_0,t_1) \in B_m} \hat{c}_{m,t_0,t_1}^{br} z_{t_0,t_1}^{m,br} \Big) \tag{1}$$

$$\sum_{t \in \{a_{j,k},\dots,f_{j,k}\}} x_{j,k,t} = 1 \quad (j,k) \in O \tag{2}$$

$$\sum_{q=0}^{t-d_{j,k}^{pr}} x_{j,k,q} - \sum_{q=0}^{t} x_{j,k+1,q} \geq 0 \quad j \in J,\ k < |O_j|,\ t \in [T] \tag{3}$$

$$\sum_{(j,k) \in O_{|m}^M} \sum_{q=\max(t-d_{j,k}^{pr}+1,0)}^{\min(t+d_{j,k}^{se},T-1)} x_{j,k,q} + z_{m,t}^{st} + \sum_{\substack{(t_0,t_1) \in B_m: \\ t \in \{t_0,\dots,t_1\}}} z_{t_0,t_1}^{m,br} = 1 \quad m \in M,\ t \in [T] \tag{4}$$

$$\sum_{(-d_m^{rd},t_1) \in B_m} z_{-d_m^{rd},t_1}^{m,br} = 1 \quad m \in M \tag{5}$$

$$\sum_{(t_0,T+d_m^{ru}-1) \in B_m} z_{t_0,T+d_m^{ru}-1}^{m,br} = 1 \quad m \in M \tag{6}$$

$$x_{j,k,t} \in \{0,1\} \quad (j,k) \in O,\ t \in [T] \tag{7}$$

$$z_{t_0,t_1}^{m,br} \in \{0,1\} \quad m \in M,\ (t_0,t_1) \in B_m \tag{8}$$

$$z_{m,t}^{st} \in \{0,1\} \quad m \in M,\ t \in [T] \tag{9}$$

The objective (1) describes the total energy cost. Equalities (2) ensure that each task is started once. Constraints (3) describe the precedence relations between consecutive tasks of each job. Equalities (4) enforce that each machine is either processing or setting up a task or in a break or standby in each period. Constraints (5) and (6) ensure each machine is offline in periods 0 and T.

3 Propagation and Presolving

In this section, we discuss several preprocessing and propagation techniques to reduce the size of the proposed integer program (1)–(9) at the root node and within the branch and bound tree. In practice, such reductions are of utmost importance to efficiently solve large models. Many techniques such as detecting dominating columns, bound tightening, and conflict analysis are implemented in general purpose ILP solvers [1]. However, these techniques heavily exploit problem-specific structures, whose (re-)detection from the model is computationally expensive and available only for some very general types of substructures. In our application, where the connection among precedence constraints and time windows plays a key role, problem-specific techniques are necessary.

Precedence constraints and time windows. Throughout this section, we denote by $a_{j,k}$ and $f_{j,k}$ the earliest and the latest period when task $(j,k) \in O$ may start its processing, respectively, at the current branch and bound node. This time window may have holes, when task (j,k) is not allowed to start processing. Pretending that there are no holes in $[a_{j,k}, f_{j,k}]$, we first apply the constraint propagation rules from [6] to detect locally valid precedence constraints and tighten $a_{j,k}$ and $f_{j,k}$.

Conflicts of breaks and tasks. Next, we try to infer implications stemming from overlaps of time windows of tasks and a single break-variable. Clearly, the break $z^{m,br}_{t_0,t_1}$ on machine $m \in M$ with $(t_0,t_1) \in B_m$ cannot participate in any integer feasible solution, if both $t_0 < a_{j,k} + d^{pr}_{j,k}$ and $t_1 \geq f_{j,k} - d^{se}_{j,k}$ hold for an arbitrary task $(j,k) \in O^M_{|m}$. This condition indicates that the break would conflict with each possible start of processing of task (j,k).

Irrelevant breaks. A break-variable $z^{m,br}_{t_0,t_1}$, $m \in M$ and $(t_0,t_1) \in B_m$, will not be used in any optimal integral solution, if there is a combination of other break- and standby-variables on machine m that exactly cover the periods from t_0 until t_1 with a smaller objective. We detect those cases by enumerating all possible covers with exactly one break. If all energy prices are non-negative, non-artificial breaks before the first and after the last task on a machine are unnecessary. Hence, break-variable $z^{m,br}_{t_0,t_1}$ with $t_0 > 0$ and $t_1 \leq T$ can be eliminated if $t_0 \leq \min_{(j,k)\in O^M_{|m}} (a_{j,k} + d^{pr}_{j,k} - 1)$ or if $t_1 \geq \max_{(j,k)\in O^M_{|m}} (f_{j,k} - d^{se}_{j,k})$.

Clique information. Modern ILP solvers automatically generate clique inequalities using a graph describing pairwise conflicts of binary variables, c.f. [1]. Not all such conflicts are detected automatically. In our code, we explicitly add conflicts for pairs of break-variables if their combined lengths exceed the duration of the overall time window T minus the sum of all task setup and processing actions, the initial ramp-up, and the final ramp-down on the machine. More explicitly, we add all conflicts between break variables $z^{m,br}_{t_0,t_1}$ and $z^{m,br}_{t_2,t_3}$ with $t_3 - t_2 + t_1 - t_0 + 2 > T + d^{ru}_m + d^{rd}_m - \sum_{(j,k)\in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k})$. In addition, the clique constraints $x_{j,k,t}+\sum_{q=t-d^{se}_{j,k}-d^{pr}_{i,l}+1}^{t+d^{pr}_{j,k}-1+d^{se}_{i,l}} x_{i,l,q} \leq 1$ for all pairs $(j,k),(i,l) \in O^M_{|m}$ and (meaningful) periods $t \in [T]$ are added to the conflict graph.

General presolving for task variables. Eventually, we apply the presolving and constraint propagation rules available in the used ILP solver, see [1], for example.

4 Branching scheme

Our branching scheme consists of two rules. Our preferred branching aims to interrupt longer intervals of consecutive fractional inactivity and forces the machine to either ramp-up or ramp-down completely. Both branches sharpen the machine profile and typically increase the dual bound. To enforce integrality of the activity of m in period t', we create two child nodes, one forcing the fractional (in)activity $f_{m,t'} := \sum_{(t_0,t_1)\in B_m : t_0 \leq t' \leq t_1} z^{m,br}_{t_0,t_1}$ to 0 and the other forcing it to 1. The machine $m \in M$ and the interval $[q_0,q_1] \in Q_m$ are chosen to maximize $(q_1 - q_0) \cdot \sum_{t=q_0}^{q_1} f_{m,t}$, where Q_m denotes the set of all consecutive fractionally inactive intervals $[q_0,q_1] \subset [T^m_+]$ on m. The chosen period $t' = (\sum_{t=q_0}^{q_1} t \cdot f_{m,t})/(\sum_{t=q_0}^{q_1} f_{m,t})$ interrupts the fractional activity of m in $[q_0,q_1]$ and forbids incomplete ramping in favorable periods.

If the first rule does not find an auspicious branching, we employ the branching of [2] to adjust the time windows of single tasks by branching on the assignment constraints (2). In contrast to [2], the task $(\hat{k},\hat{j}) \in O$ to branch on is chosen to maximize $(r(\hat{j},\hat{k}) - l(\hat{j},\hat{k})) \cdot \left(\sum_{t=l(\hat{j},\hat{k})}^{r(\hat{j},\hat{k})} \sum_{(j,k)\in O^M_{|m}} x_{j,k,t}\right)^{-1}$, with $l(j,k) = \arg\min_{t\in[T]}\{x_{j,k,t} > 0\}$ and $r(j,k) = \arg\max_{t\in[T]}\{x_{j,k,t} > 0\}$. This modification prefers tasks that currently have small overlap with others, and so branching on those tasks has a strong effect on the machine activity.

5 Results

In this section, we compare results obtained by applying the introduced techniques in our branch-and-cut code in SCIP [4] (with GUROBI 9.5.1 [8]), using them to presolve the model solved by GUROBI, and using GUROBI as a standalone solver with default setting and aggressive presolving, but without our presolving methods.

The instances presented here, with real energy prices from March 2021 of Germany/Luxembourg, are derived from the instance la01 [10] by dividing all processing times by 10 and ceiling to get processing and setup times and a manageable time window ($T = 120$). Ramping times are chosen as $\frac{2}{3}$, 1, or 1.5 times the mean processing on each machine to derive instances with small (s), medium (m) and large (l) ramping durations, respectively. The energy demands are chosen as $(D^{off}_m, D^{ru}_m, D^{se}_m, D^{pr}_m, D^{st}_m, D^{rd}_m) = (0, 10, 5, 8, 3, 6)$. The resulting full models for s,m, and l have 31,198, 30,998, and 30,698 variables, respectively. The table shows the number of variables after presolve (v), the number of branch and bound nodes (NN), the primal-dual gap, and the dual bound after 3600s.

Table 1. Comparison of the effect of our propagation and branching rules

instance	la01 s				la01 m				la01 l			
	v	NN	gap	dual	v	NN	gap	dual	v	NN	gap	dual
scip+pre+bra	12k	1.2k	0.00	124291	9k	591	0.00	132448	5k	6k	0.00	146503
gurobi+pre	14k	108k	0.00	124291	9k	9k	0.00	132448	6k	113k	0.00	146503
gurobi	19k	157k	0.16	124098	23k	42k	0.05	132387	12k	164k	0.26	146162

The results show that our techniques detect more reductions even than the aggressive presolving of Gurobi. Only with our reductions the problem could be solved to optimality within the time limit. Furthermore, our tailored branching substantially reduced the number of branch-and-bound nodes that have been explored. With our brachning and presolving Scip was able so solve the problems as fast Gurobi with only our presolving and its default branching strategies, despite running single-threaded versus Gurobi using up to 8 parallel threads.

6 Conclusion

We developed preprocessing and branching techniques that enable us to solve time-indexed ILP formulations of job shop scheduling problems involving time windows, machine states and energy costs to optimality. Our reduction and propagation techniques outperform those implemented in standard ILP solvers and can also be applied easily within the branch-and-bound tree. Our experiments show that, in combination with tailored branching strategies, these techniques effectively reduce the number of variables, drive the dual bound and substantially reduce the size of the branch-and-bound trees, especially for instances with relatively large ramping-durations. From our point of view, there is still some need to improve the bounds on the task variables to obtain stronger dual bounds and derive better primal solutions faster.

Acknowledgements This version of the contribution has been accepted for publication after peer review, but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: `https://dx.doi.org/10.1007/978-3-031-24907-5_68`. Use of this Accepted Version is subject to the publisher's Accepted Manuscript terms of use `www.springernature.com/gp/open-research/policies/accepted-manuscript-terms`.

References

1. Achterberg, T., Bixby, R.E., Gu, Z., Rothberg, E., Weninger, D.: Presolve reductions in mixed integer programming. INFORMS Journal on Computing 32(2), 473–506 (2020)
2. van den Akker, J.M., et al.: A polyhedral approach to single-machine scheduling problems. Mathematical Programming 85(3), 541–572 (1999)

3. Benedikt, O., Sucha, P., Modos, I., Vlk, M., Hanzálek, Z.: Energy-aware production scheduling with power-saving modes. In: Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 72–81. Springer (2018)
4. Bestuzheva, K., et al.: The SCIP Optimization Suite 8.0. ZIB-Report 21-41, Zuse Institute Berlin (2021)
5. Bley, A., Linß, A.: Job shop scheduling with flexible energy prices and time windows. In: Operations Research Proceedings 2019, pp. 207–213. Springer (2020)
6. Brucker, P.: Scheduling and constraint propagation. Discrete Applied Mathematics 123(1), 227–256 (2002)
7. Gao, K., Huang, Y., Sadollah, A., Wang, L.: A review of energy-efficient scheduling in intelligent production systems. Complex & Intelligent Systems 6(2), 237–249 (2020)
8. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2022), `https://www.gurobi.com`
9. Jain, A., Meeran, S.: Deterministic job-shop scheduling: Past, present and future. European Journal of Operational Research 113(2), 390 – 434 (1999)
10. Lawrence, S.: Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement). Tech. rep., Graduate School of Industrial Administration, Carnegie-Mellon University (1984)
11. Nolde, K., Morari, M.: Electrical load tracking scheduling of a steel plant. Computers & Chemical Engineering 34(11), 1899–1903 (2010)
12. Selmair, M., Claus, T., Herrmann, F., Bley, A., Trost, M.: Job shop scheduling with flexible energy prices. In: Proceedings of the 30th ECMS (2016)
13. Shrouf, F., Ordieres-Meré, J., García-Sánchez, A., Ortega-Mier, M.: Optimizing the production scheduling of a single machine to minimize total energy consumption costs. Journal of Cleaner Production 67, 197–207 (2014)
14. Zhang, J., Ding, G., Zou, Y., Qin, S., Fu, J.: Review of job shop scheduling research and its new perspectives under industry 4.0. Journal of Intelligent Manufacturing 30(4), 1809–1830 (2019)