Job-shop scheduling with flexible energy prices and time windows: A branch-and-price-and-cut approach

By

Andreas Linß

A thesis submitted for the
degree of Doktor der Naturwissenschaften (Dr. rer. nat.)

in the

Faculty of Mathematics and Natural Sciences
University of Kassel

Date of submission: December 18, 2023
Disputation date: April 11, 2024

First Reviewer: **Prof. Dr. Andreas Bley**
Second Reviewer: **Prof. Dr. Marc Pfetsch**

Kassel, June 17, 2024.

Abstract

Energy-aware scheduling is crucial in the current economy and green production initiatives. However, with the rise of renewable energy sources, power production is subject to uncertain weather conditions. Hence, within a network, some balancing group managers must maintain a balance between production and demand, which requires precise energy orders for specific times. Therefore, those managers must prioritize accurate energy orders to manage this complex problem.

The primary aim of this thesis is to manage one customer's energy order for a series of energy-dependent tasks efficiently. It is crucial to recognize that energy costs are subject to fluctuations and that the customer's primary concern is to minimize costs, assuming all orders are completed.

To embed the customers' interests into a mathematical model, we consider the job-shop scheduling problem with time windows, precedence constraints and machine states, where the challenge is to organize the energy required to process a set of jobs. We aim to create a feasible processing start for each task while ensuring that the machines consume the least amount of energy. To that end, we use an integer programming formulation that couples the scheduling of tasks with the assignment of the corresponding machine states. The objective function is the price of the consumed energy. It turns out that the considered scheduling problem is NP-hard in general. Nevertheless, we present some relevant cases that are solvable in polynomial time. Scheduling problems are notoriously difficult due to the intertwining of time-based events and job processing start times. To address this, we use a time-indexed formulation using many variables. Additionally, proving the optimality of primal solutions becomes increasingly time-consuming as the problem size grows. To handle a huge number of variables, we use presolving rules specifically designed for the problem and involve combinatorial conditions to reduce the number of variables quickly. Those presolving rules can also be used as constraint propagation rules within the branch-and-bound tree. For faster problem-solving, we consider problem-adapted branching rules, constraints, and heuristics. This work highlights that the classical scheduling techniques are inefficient when considering energy prices since they are designed to schedule the tasks as early as possible. Therefore, new techniques need to be developed that involve the total energy cost when scheduling the tasks to different periods. To get an efficient branch-and-bound algorithm, near-optimal primal bounds within the early stages are required. Therefore, classical list scheduling heuristics and large neighborhood search algorithms are used to compute near-optimal primal solutions. However, these algorithms do not consider the objective in the first place. To address this issue, a dynamic programming approach is implemented to compute the optimal solution for a fixed order of jobs. Additionally, a neighborhood search is implemented, which reuses the dynamic program to evaluate candidates within the neighborhood. Techniques such as diving heuristics and genetic algorithms are also employed to find early, near-optimal primal solutions. As a result, we obtain a problem-adapted dual bound driving solution approach, which is able to compute near-optimal solutions and dual bounds efficiently.

Acknowledgments

This work would not have been possible without the support of numerous individuals.

I extend my sincere gratitude to Professor Andreas Bley for his exceptional guidance, support, and invaluable mentorship throughout the completion of this thesis. His expertise, constructive feedback, and encouragement have played a pivotal role in shaping this work and its implementations.

I also express my heartfelt appreciation to my esteemed colleagues at the University of Kassel, whose support and camaraderie have been indispensable throughout this journey. The collaborative spirit and intellectual exchange within our team have enriched my understanding and provided valuable perspectives.

Special thanks to Thomas Izgin and Philipp Hahn for their insightful discussions and willingness to share their expertise. The positive and collaborative environment at the University of Kassel significantly contributed to the completion of this academic endeavor.

My sincere gratitude extends to Arthur Linß and Ilija Afanasyev for their meticulous proofreading of this thesis. Their keen attention to detail and insightful suggestions have greatly enhanced the clarity and precision of the thesis. I am thankful for their dedicated efforts in ensuring the accuracy of the language and formatting, undoubtedly elevating the quality of this work. Their commitment to excellence has had a positive impact on the final presentation of this thesis. I would also like to thank my parents, whose support and encouragement made the path to write this thesis possible in the first place. Last but most importantly, I want to express my heartfelt gratitude to my wife, Carolin. Her unwavering support and encouragement were my anchors throughout this PhD journey. I am profoundly thankful for her sacrifices and belief in my abilities, which made this academic endeavor not only possible but immensely fulfilling. To Carolin, my greatest supporter, thank you.

Contents

Chapter 1

Introduction

We consider the job-shop scheduling problem with flexible energy prices and time windows. The job-shop scheduling problem is an NP-hard combinatorial optimization problem. A well-known fact is that switching the objective function can increase or decrease the complexity of scheduling problems. Also, adding precedence constraints, which decrease the number of feasible solutions, can increase or decrease the complexity of the considered scheduling problem. We consider the sum of weighted energy consumption to be the objective. This objective requires the computation of associated machine states, as well as their coupling to the processing starts of the respective tasks. The formulation of the tasks' scheduling and the machine states' modeling to compute the objective function highly influences the efficiency of the resulting solution approach. Scheduling problems can be formulated as integer programming problems and solved with commercial solvers. Often, commercial solvers require many branch-and-bound nodes and a lot of time to prove the computed solutions' optimality. This thesis presents an approach which uses less branch-and-bound nodes to compute the optimal solution in less time as one selected commercial solver.

1.1 Focus of This Thesis

The job-shop scheduling problem is one of the hardest combinatorial optimization problems. This thesis describes the development of a problem-specific branch-and-cut, as well as a branch-and-price algorithm to solve the job-shop scheduling problem with flexible energy prices and time windows. The problem can be formulated using integer programming, and commercial solvers can solve this problem formulation. However, commercial solvers are implemented to solve a variety of optimization problems efficiently and therefore many properties of scheduling problems are not exploited. To that end, we first analyze the problem to learn about its special properties. Using the results of this analysis leads to a problem-adapted integer programming formulation. In particular, the analysis of fractional solutions reveals that well-known techniques of classical scheduling problems are misleading. Thus, we develop, present and implement well-known and new algorithms and techniques within the problem-specific setting. In addition to the set of algorithms, the branching, separating and heuristic algorithms are developed and implemented so that we profit from the collaboration of the different techniques. The arising subproblems will be discussed, and at the end of the thesis, we will obtain an algorithm that determines near-optimal solutions quickly and outperforms commercial solvers using the same number of threads.

1.2 Overview of the Thesis Structure

This thesis is divided into four major parts: the introduction of the problem and the discussion of its complexity in Section 2, the presentation of integer programming formulations and the comparison of the descriptions of the feasible solution spaces in Chapter 3, the presentation of the solution approach and the implemented algorithms in Chapter 4, as well as the computational experiments in Chapter 5. This work closes with a summary and a conclusion.

In the first part, we present the problem in Section 2.1 followed by the substantiation of the problem's relevance. This section is followed by a presentation of the related literature in Section 2.5 and a categorization of the considered research topic. This part closes with the analysis of the problem's complexity in Section 2.6, which also justifies the intensified study of the problem.

The second part commences with the presentation of the model in Section 3.1: a straightforward time-indexed problem formulation will be presented, followed by a partial Dantzig–Wolfe reformulation. Both formulations are integer linear programs (ILP), respectively mixed-integer linear programs (MILP). Various valid inequalities are presented to improve the description of the feasible solutions. In addition, inequalities, which cut off non-optimal solutions, are considered to accelerate the solving process. We compare the Dantzig–Wolfe reformulation and the straightforward time-indexed formulation to show that the partial Dantzig–Wolfe reformulation leads to a tighter description of the integral feasible solutions.

The third part deals with the problem-solving process and the strengthening of the linear programming formulation relaxation at each branch-and-bound node. We start with the reduction of the problem size in Section 4.1. The classical presolving techniques, like dominating columns and probing, are briefly reviewed. Then, the presolving reduction is transferred into a problem-specific combinatorial counterpart. These counterparts have been assigned to combinatorial optimization problems, and efficient solution techniques are proposed and presented. The next Section 4.2 summarizes the consequences of classical scheduling-related branching rules and our implemented branching rules. Moreover, we justify the implementation of a new branching algorithm, creating equally strong branches. Additionally, the implemented branchings are designed so that the propagation algorithms (former presolving rules) detect more domain reductions. In general, the branch-and-bound nodes are solved by linear programming (LP). The linear programming relaxation of branch-and-bound nodes is strengthened by additional valid inequalities called cutting planes. Section 4.3 includes the presentation of problem-specific cutting planes. We consider the known general upper bound (GUB) cover constraints for single-machine scheduling and extend them to be strong within our problem setting. Moreover, we derive conflicts from combinatorial substructures and add them manually to the conflict graph, which is used to derive clique cuts. This section closes by presenting valid constraints derived from the linear ordering subproblem of the single-machine scheduling problem. A lifting scheme is proposed to strengthen those inequalities within our problem setting. Implementing a column-generation algorithm is natural since we proposed a Dantzig–Wolfe reformulation. We present in Section 4.4 the different algorithms we implemented to solve the pricing problems. As mentioned before, branch-and-bound algorithms are more efficient if we can compute near-optimal primal solutions earlier. We then proceed in Section 4.5 with the description of the implemented heuristics, which include list scheduling heuristics, genetic algorithms and local search approaches. In addition, we developed a dynamic programming approach that can be used to compute the optimal solution for a fixed execution order of the tasks.

This thesis closes with the fourth part describing the implementation in C++ and the choice of parameters in Chapter 5. Furthermore, computational experiments are presented and analyzed to demonstrate the implemented methods' and heuristics' efficiency. Finally, we give a summary of the implemented methods and the experimental results. Moreover, we provide a brief outlook on further approaches for future research work.

1.3 Main Contributions

The main contributions of this thesis are the following.

1. We provide a (partial) Dantzig–Wolfe reformulation of a time-indexed formulation for an energy-aware job-shop scheduling problem. This reformulation can be regarded as a set partitioning problem with precedence constraints and is proven to provide a better description of the machine transitions. The number of variables thereby increases only polynomially in the size of the variables of the original formulation. In particular, our partial Dantzig–Wolfe reformulation introduces variables representing periods of ramping. Ramping is the change of machine state from offline to online or from online to offline within at least one period. The reformulation is described in [BL20].

2. We provide problem-specific presolving techniques using conflict analyses and known presolving rules. The latter are encoded in combinatorial conditions that could be checked more easily than the steps in classical preprocessing rules. The presolving rules are presented in [BL23].

3. We present specifically designed branching schemes and a branching selection rule to explore the solution space efficiently. Those branching rules are variants of the special ordered set (SOS)-branching. We propose a new and descriptive way of computing the branching disjunction in the case of job-shop scheduling with flexible energy prices and time windows. Moreover, we extensively analyze the fractional solutions of the considered optimization problem.
4. We devise a dynamic program to solve the job-shop scheduling problem in the case of a fixed order of the tasks on each machine. We embed the dynamic program into a large neighborhood search and obtain the possibility to solve the optimization problem by investing a lot of time.

Chapter 2

Problem Description and Notation

In this chapter, we will introduce the formal notation of the problem formulation. Then, in addition to the classical parameters of the job-shop scheduling problem, new attributes such as machine states, different energy requirements and energy costs will be introduced. Subsequently, a feasible problem solution is formally described. Then, we present some background information on the electricity market, which provides energy prices and helps to define the objective function of our considered optimization problem. After that, the research-related literature is reviewed, followed by some relevant results concerning the computational complexity of the problem.

2.1 Formal Notation and Problem Setting

To describe the problem, we use a special notation for intervals of integral numbers.

Definition 2.1.1. *Let $n \in \mathbb{N}$. The set, including the first n nonnegative integral numbers, is defined by*

$$[n[_{\mathbb{Z}} := \{0, \ldots, n-1\}.$$

Moreover, we are also interested in specifying the left bound of the numbers.

Definition 2.1.2. *Let $n, m \in \mathbb{N}$. The set including the integral numbers between inclusively m and exclusively n, is defined by*

$$[m, n[_{\mathbb{Z}} := [n[_{\mathbb{Z}} \setminus [m[_{\mathbb{Z}} = \{m, \ldots, n-1\}.$$

In the case of the job-shop scheduling problem with flexible energy prices and time windows, the planning horizon $[T[_{\mathbb{Z}} := \{0, \ldots, T-1\}$ consists of T uniform time periods. For each period $t \in [T[_{\mathbb{Z}}$, we are given a time-indexed weight $P_t \in \mathbb{Q}$. The weight P_t is valid during period t and represents the energy price in that period. In addition, we are given a set of $n_M \in \mathbb{N}$ (non-)uniform machines, denoted by $M := [n_M[_{\mathbb{Z}}$, and a set of jobs $J := [n_J[_{\mathbb{Z}}$, $n_J \in \mathbb{N}$. Each job $j \in J$ is associated with a list $O^J_{|_j}$ of $n_j \in \mathbb{N}$ tasks, defined by $O^J_{|_j} := \{(j, 0), \ldots, (j, n_j-1)\}$. The pairwise distinct tasks $(j, k), (j, l) \in O^J_{|_j}$ have to satisfy a precedence order. The order is defined as follows: task (j, k) has to precede task (j, l) iff $k < l$. The precedence relation of two pairwise distinct tasks is described by $(j, k) \to (j, l)$ and denotes that the preceding task (j, k) needs to complete its processing before the succeeding task (j, l) can start its processing. Each task (j, k) must be set up and processed on a predefined machine $m_{j,k} \in M$. Note that the setup and the processing are completed on the same machine. It is assumed that the processing has to immediately follow up on the setup. To summarize the set of all tasks of a given problem, we introduce the set $O := \bigcup_{j \in J} O^J_{|_j}$. To easily describe all tasks that need to be processed on machine $m \in M$, we introduce the set $O^M_{|_m} = \{(j, k) \in O \mid m_{j,k} = m\}$. For each task $(j, k) \in O$, we are given its setup duration $d^{se}_{j,k} \in \mathbb{N} \cup \{0\}$ and its processing duration $d^{pr}_{j,k} \in \mathbb{N}$.

In addition, we are given a release date $a_j \in [T[_{\mathbb{Z}}$ and a due date $f_j \in [T[_{\mathbb{Z}}$ for each job $j \in J$, which apply to the first and the last task of the job, respectively. The first task $(j, 0)$ of job $j \in J$ can only start processing in or after period a_j, but the setup of task $(j, 0)$ can start before period a_j. The last task (j, k) of job $j \in J$ can only start processing after all its predecessors have completed their processing. Moreover, the task (j, k) must finish processing before period f_j. Therefore, the last processing start of task (j, k) is period $f_j - d^{pr}_{j,k}$.

One constraint of the problem is that in each period $t \in [T[_{\mathbb{Z}}$, each machine $m \in M$ must be in exactly one of the operating states *off*, *processing*, *setup*, *standby*, *ramp-up* or *ramp-down*, summarized as $\mathtt{S} := \{\mathit{off}, pr, se, st, ru, rd\}$. A machine is called *active* if its operating state is *setup*, *processing*, or *standby*. Otherwise, it is called *inactive*.

The machine is not allowed to switch between its states arbitrarily. A feasible machine-state transition must follow the rules:

1. If the machine is running in machine state *off* in period $t \in [T - 1[_{\mathbb{Z}}$, then the machine can be *off* or in state *ramp-up* in period $t + 1$.
2. If the machine is running in machine state *ramp-up* in period $t \in [T - 1[_{\mathbb{Z}}$, then the machine can be in any state $s \in \{\textit{ramp-up}, \textit{standby}, \textit{setup}, \textit{processing}, \textit{ramp-down}\}$ in period $t + 1$.
3. If the machine is running in machine state *standby* in period $t \in [T - 1[_{\mathbb{Z}}$, then the machine can be in any state $s \in \{\textit{standby}, \textit{setup}, \textit{processing}, \textit{ramp-down}\}$ in period $t + 1$.
4. If the machine is running in machine state *setup* in period $t \in [T - 1[_{\mathbb{Z}}$, then the machine can be in any state $s \in \{\textit{setup}, \textit{processing}\}$ in period $t + 1$.
5. If the machine is running in machine state *processing* in period $t \in [T - 1[_{\mathbb{Z}}$, then the machine can be in any state $s \in \{\textit{standby}, \textit{setup}, \textit{processing}, \textit{ramp-down}\}$
6. If the machine is running in machine state *ramp-down* in period $t \in [T - 1[_{\mathbb{Z}}$, then the machine can be in any state $s \in \{\textit{ramp-up}, \textit{off}, \textit{ramp-down}\}$ in period $t + 1$.

The transition to another machine state is only valid if the switch is feasible and no setup, processing, or ramping is interrupted prematurely. The duration of the *ramp-up* phase, changing from *off* to any state $s \in \{se, pr, st, rd\}$, is $d^{ru}_m \in \mathbb{N}$. Analogously, the duration of the *ramp-down* phase is $d^{rd}_m \in \mathbb{N}$. We explicitly prohibit instances with $d^{rd}_m = 0$ or $d^{ru}_m = 0$. A ramping duration of zero periods needs to be described by another set of rules, which is not part of this thesis. The minimum duration of the ramping is 1 to ensure that the switching rules can be satisfied.

Moreover, the parameter $D^s_m \in \mathbb{Q}$ denotes the energy demand of machine $m \in M$ in state $s \in \mathtt{S}$. Using the energy demand, the direct ramping can be approximated by allowing an energy demand of 0 for the ramping.

From the release and due dates, the precedence constraints of the job sequences and the ramping duration, we can derive the implied allowed processing starts for each task $(j, k) \in O$, with the placeholder $a_{j,-1} = a_j$, as

$$a_{j,k} = \max\left(a_{j,k-1} + d^{pr}_{j,k-1}, d^{ru}_{m_{j,k}} + d^{se}_{j,k}\right)$$

and

$$f_{j,k} = \min(f_j, T - d^{rd}_{m_{j,k}}) + 1 - \sum_{q=k}^{|O_j|} d^{pr}_{j,q}.$$

The allowed processing starts can be summarized by $\mathcal{A}_{j,k} := \{a_{j,k}, \ldots, f_{j,k}\}$.

The machine is assumed to be *offline* at the beginning of the time window as well as at the end of the time window. However, the machine can start the ramp-up in period 0 and use the period $T - 1$ to finish the ramp-down.

Summary of a Feasible Solution

A feasible solution consists of the processing start for each task 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 on the same machine if the machine can complete the setup in time. In contrast, the predecessor is processing (on another machine). The start of the first task 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 the state *processing* or *setup*, respectively. Otherwise, the machine can be active in *standby* or become inactive (*ramp-down*, *off*, or *ramp-up*) while respecting the ramping durations and state switching rules mentioned above. Each machine must be *off* at the planning horizon's beginning and end.

Nevertheless, the machine can start the *ramp-up* in period 0 and finish the *ramp-down* in period $T-1$. We aim to find a schedule of tasks with minimized energy costs. To describe the feasible solution, we use an extended time window $[T_+[_\mathbb{Z} = [T[_\mathbb{Z} \cup \{-1, T\}$.

Definition 2.1.3 (Feasible solution)**.** *The tuple $(\mathcal{S}^J, \mathcal{S}^M)$, with*

$$\mathcal{S}^J : O \to [T[_\mathbb{Z}$$

and

$$\mathcal{S}^M : M \times [T_+[_\mathbb{Z} \to \mathtt{S}$$

is called a ***feasible solution of the job-shop scheduling problem with flexible energy prices*** *if the following conditions hold:*

1. $\mathcal{S}^J(j,k) + d^{pr}_{j,k} \leq \mathcal{S}^J(j,k+1)$ *for all* $(j,k),(j,k+1) \in O$.
2. $\mathcal{S}^J(j,k) \in [a_{j,k}[_\mathbb{Z}\ f_{j,k}$ *for all* $(j,k) \in O$.
3. *For all* $m \in M$ *and* $t \in [T[_\mathbb{Z}$ *the condition* $\mathcal{S}^M(m,t) = pr$ *must hold, if the period* t *satisfies* $t \in [\mathcal{S}^J(j,k), \mathcal{S}^J(j,k) + d^{pr}_{j,k}[_\mathbb{Z}$.
4. *For all* $m \in M$ *and* $t \in [T[_\mathbb{Z}$ *the condition* $\mathcal{S}^M(m,t) = se$ *must hold, if the period* t *satisfies* $t \in [\mathcal{S}^J(j,k) - d^{se}_{j,k}, \mathcal{S}^J(j,k)[_\mathbb{Z}$.
5. $\mathcal{S}^J(j,k) + d^{pr}_{j,k} + d^{se}_{i,l} \leq \mathcal{S}^J(i,l)$ *or* $\mathcal{S}^J(i,l) + d^{pr}_{i,l} + d^{se}_{j,k} \leq \mathcal{S}^J(j,k)$ *holds for all distinct* $(j,k),(i,l) \in O^M_{|m}$ *and* $m \in M$.
6. $\mathcal{S}^M(m,-1) = \mathcal{S}^M(m,T) = \mathit{off}$ *for each* $m \in M$.
7. *If* $\mathcal{S}^M(m,t) = \mathit{off}$ *and* $\mathcal{S}^M(m, t + d^{ru}_m + 1) \in \{st, pr, se, rd\}$, *then* $\mathcal{S}^M(m,q) = ru$ *for* $q \in [t+1, t+d^{ru}_m+1[_\mathbb{Z} \subseteq [T_+[_\mathbb{Z}$.
8. *If* $\mathcal{S}^M(m,t) \in \{st, pr, se, rd, \mathit{off}\}$ *and* $\mathcal{S}^M(m, t + d^{rd}_m + 1) = \mathit{off}$, *then* $\mathcal{S}^M(m,q) = rd$ *for* $q \in [t+1, t+d^{rd}_m+1[_\mathbb{Z} \subseteq [T_+[_\mathbb{Z}$.
9. *For each* $m \in M$ *and* $t \in [T[_\mathbb{Z}$ *the condition* $\mathcal{S}^M(m,t) \in \{\mathit{off}, rd, ru, st\}$ *must hold for* $t \notin \bigcup_{(j,k)\in O^M_{|m}} \{\mathcal{S}^J(j,k) - d^{se}_{j,k}, \ldots, \mathcal{S}^J(j,k) + d^{pr}_{j,k} - 1\} \subseteq [T_+[_\mathbb{Z}$.
10. *If* $\mathcal{S}^M(m,t) = s$ *for* $s \in \{rd, ru\}$, *then there exists a period* $t_0 \in [T]$ *and a subset* $\{t_0, \ldots, t_0 + d^s_m - 1\} \subseteq [t - d^s_m, t + d^s_m[_\mathbb{Z}$ *with* $\mathcal{S}^M(m,q) = s$ *for all* $q \in [t_0, t_0 + d^s_m[_\mathbb{Z} \subseteq [T_+[_\mathbb{Z}$.

The mapping $\mathcal{S}^J$ describes the tasks' processing starts, and the mapping $\mathcal{S}^M$ denotes the assignment of the periods to the machine states for each machine.

Suppose we are given a large time window and a feasible task schedule $\mathcal{S}^J$. Then, for example, the machine could run in state standby for an arbitrary number of periods until the machine finally ramps down. Thus, several valid machine state assignments are feasible for the given task schedule $\mathcal{S}^J$.

Lemma 2.1.4. *Let $\mathcal{S}^J$ denote a feasible scheduling of the tasks. Then, the corresponding feasible machine state assignment $\mathcal{S}^M$ is generally not unique.*

The machine state assignment $\mathcal{S}^M$ of a single machine is visualized in Figure 2.1.

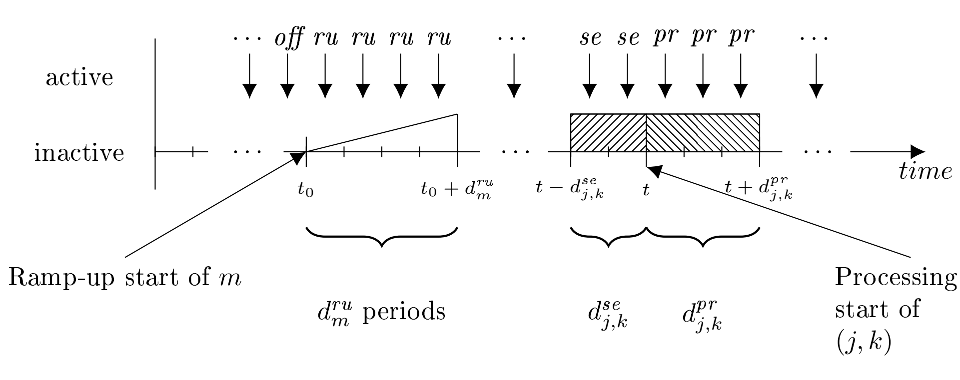


Figure 2.1: Visualization of ramping, setup, and processing durations, and the corresponding machine states. This visualization demonstrates that the start of the ramp-up in period t_0 blocks the machine in the following d^{ru}_m periods. In period $t_0 + d^{ru}_m$, the machine is active and must switch to another machine state. The processing start of a task $(j,k) \in O^M_{|m}$ blocks the machine from period $t - d^{se}_{j,k}$ until the start of period $t + d^{pr}_{j,k}$. The machine finishes the processing at the end of period $t + d^{pr}_{j,k} - 1$ and is allowed to switch the state in period $t + d^{pr}_{j,k}$.

2.2 Summary of the Problem Parameters

Altogether, the problem parameters can be stated as follows:

Name: Job-shop scheduling with flexible energy prices and time windows.

Goal: Compute a feasible schedule $\mathcal{S}^J$, and the corresponding machine states $\mathcal{S}^M$, such that the energy costs for ramping, standby, setup, and processing are minimal.

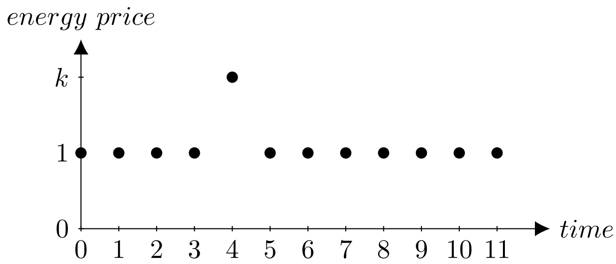
Problem parameters

- A set of machines M.
- The set of tasks O
- A mapping $m_{j,k}$ of the tasks $(j,k) \in O$ to the machines $m \in M$.
- A release and a due date for each job $j \in J$.
- A set of precedence relations between the tasks $(j,k) \in O^J_{|j}$ for each $j \in J$.
- A time window $[T[_{\mathbb{Z}}$.
- A processing duration $d^{pr}_{j,k} \in \mathbb{N}$ and a setup duration $d^{se}_{j,k} \in \mathbb{N} \cup \{0\}$ for each task $(j,k) \in O$.
- Ramping durations $d^{rd}_m \in \mathbb{N}$ and $d^{ru}_m \in \mathbb{N}$ for each $m \in M$.
- An energy demand $D^s_m \in \mathbb{Q}$ for each machine m and each machine state $s \in \mathtt{S}$.
- An energy price P_t for each period $t \in [T[_{\mathbb{Z}}$.

2.3 Relevance of This Problem

The problem of computing a schedule with minimal energy costs is not the same as computing a schedule with a minimum makespan. The minimum makespan optimization compute a schedule that minimizes the time from the start of setup on any machine to the completion of the last process on any machine. The following example shows that a shorter schedule does not automatically mean lower energy costs.

Example 2.3.1. *We consider the job-shop scheduling problem with three jobs, each with two tasks. All processing and setup durations are equal to* 1*. The time window is set to* $T = 11$*. We are given two machines with ramping durations* $d^{rd}_m = d^{ru}_m = 1$ *and* $D^s_m = 1$ *for each* $m \in M$ *and* $s \in \mathtt{S} \setminus \{\mathit{off}\}$*. There are no task-depending time windows, so the release and due dates of the jobs are* $a_j = 0$ *and* $f_j = T$*. The first task always starts on machine* 1*, and the second task of each job sequence starts processing on machine* $m = 2$*. The energy prices are chosen to always be* 1 *except in the period of* 4*. We fix the energy costs to some large value* $k \in \mathbb{Z}$*. The optimal solution of a schedule regarding the minimum makespan*



is shown in Figures 2.2 and 2.3. A ramp-up of length 1 *is represented here by a change from 0 to 1 within one period. A ramp-down of length* 1 *is visualized by a switch from* 1 *to* 0 *from period* 7 *to period* 8 *on machine* 1*. A delimited block, like the one from* $t = 1$ *to* $t = 3$ *on machine* 1*, represents the setup and processing of a single task. We assume that the tasks are processed in order according to their job ID.*

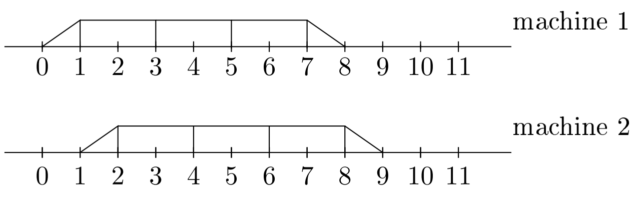


Figure 2.2: Solution of the makespan optimization.

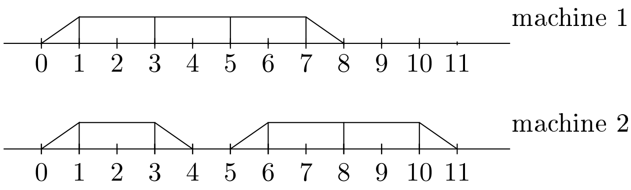


Figure 2.3: Solution of the energy price optimization.

Shifting the makespan solution to the best location does not affect the objective value since either the processing end or the processing start of the complete schedule is scheduled to the expensive period of 4. *Therefore, the optimal objective value of the makespan solution is always* $7 \cdot 1 + 7 \cdot 1 + 2 \cdot k = 2 \cdot k + 14$. *In contrast, we present a feasible solution to the job-shop scheduling problem with flexible energy prices in Figure 2.3. This solution uses an additional ramp-down and ramp-up on machine* 2. *Thus, the machine can save the energy price of the period* 4 *once. The objective of this solution is* $7 \cdot 1 + k + 9 \cdot 1 = 16 + k$. *Therefore, the objective of the energy-price optimized solution will always be cheaper for* $k > 2$. *It is worth mentioning that the makespan solution would also be an optimal solution for* $k \leq 2$ *and would be computed by the energy-aware scheduling problem.*

2.4 Remarks on the Electricity Market

The considered objective function is the minimization of the weighted consumption of energy. The weights of the energy consumption are considered to be the energy prices of the electricity market. To describe the realization of those energy prices and the knowledge about the costs of each period, a short review of the functionality of the electricity market of Europe is proposed.

The electricity market in Europe is called *European Power Exchange (EPEX SPOT) SE*. The market is based in Paris and has other offices in many European capitals. The European electricity market allows different companies to organize the production and supply of electricity and the trading, marketing, and transmission. Due to the lack of batteries, electricity cannot be stored, and thus, supply and demand should always be balanced. Furthermore, power losses need to be considered due to long transport distances. Moreover, the operational failures of producers or transmission lines, which cause reduced energy production, need to be considered in real-time. The electricity market ensures balanced demand and production to maintain the safety of the network.

The market participants can order electricity at various markets, which differ in delivery time. At the day-ahead market, once per day, there will be a blind auction. All hours of the next day will be treated within the same auction. The participants send two offers for each hour: one that includes their demand or supply of energy for each period of the next day and an offer that provides block orders and links the delivery periods. The power exchange price is determined as the market-clearing price from the offers and bids of energy prices and volume. Buyers who accept to buy for a higher price will pay the market-clearing fee. The sellers who accept to sell for a lower price will sell for the market-clearing price.

On the intraday market, the participants trade electricity that must be delivered within the same day. The distance between trade and delivery can go down to five minutes. The participants use the intraday market for adjustments in supply production to adapt their supply or demand to real-time data. If the energy production or supply is not balanced, compensation energy needs to be used to withdraw or give energy to the grid at a high price. The compensation energy is provided by companies and by energy producers that can increase or decrease their production within seconds. In addition to long-term contracts for buying a fixed part of the energy, renewable energy strongly impacts the prices. For example, storms and sunny days may lead to low energy prices, while rainy or wind-free days can lead to high prices. Figure 2.4 illustrates the advantage of optimizing the energy demand a day ahead. Moreover, Figure 2.5 depicts an example of an objective function the customers would need to optimize daily.

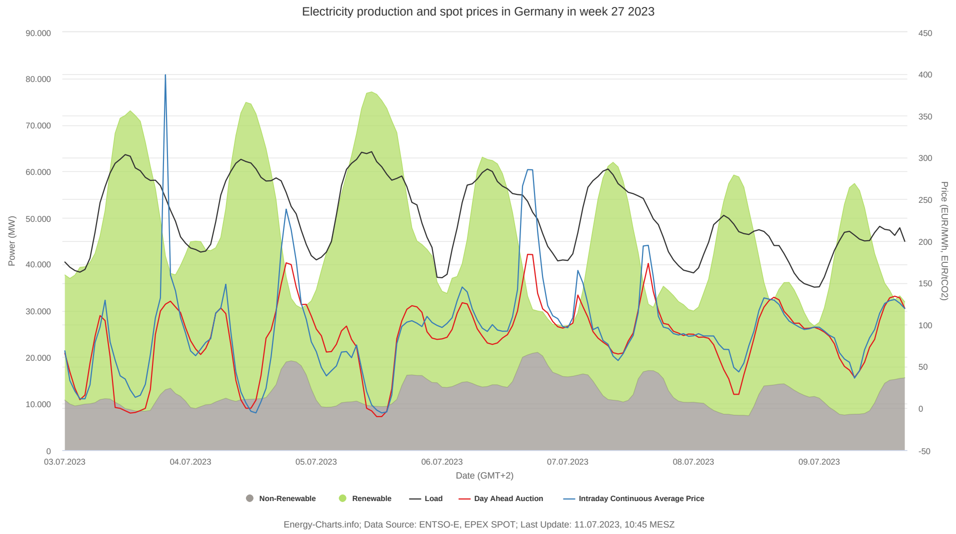


Figure 2.4: The graphic shows the energy prices of the market area of Germany and Luxembourg from July 3, 2023, to July 10, 2023. The green area depicts the amount of renewable power within the network. The gray area describes non-renewable power. Renewable energy changes within a day-night rhythm. In addition, the 27th week was really hot and sunny. The red line describes the energy price of the day-ahead auction. The blue line represents the intraday auction price, which must be paid to balance the power demand. One can also see that the energy price decreases when the amount of renewable power increases. Moreover, one can observe on the evening of July 3, 2023, that the intraday market price can significantly increase if the energy demand surpasses the amount of provided energy.

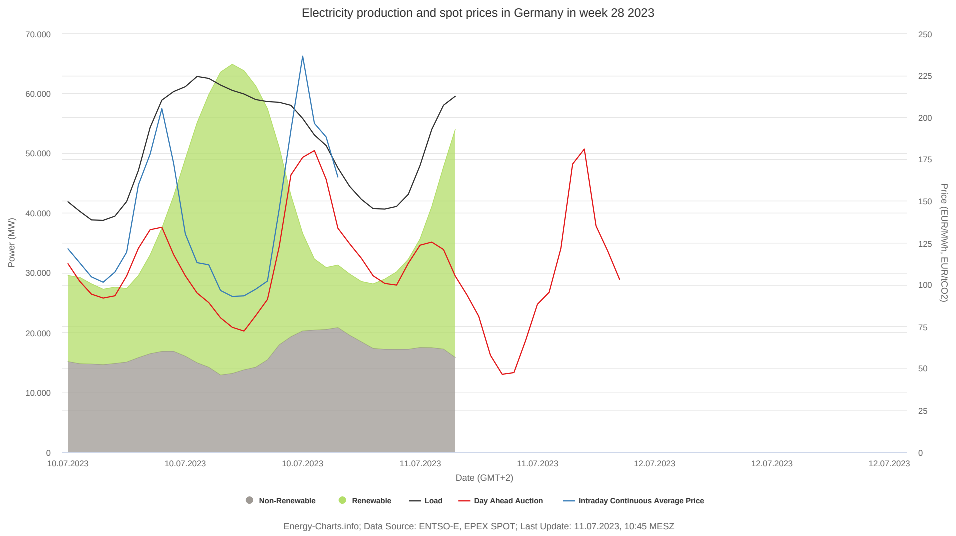


Figure 2.5: The graphic shows the energy prices of the market area of Germany and Luxembourg from July 10, 2023, up to July 12, 2023. We see about an energy-market customer on July 11, 2023. Moreover, the energy price (red line) for the next 24 hours is known. Thus, the customer can use this information to schedule the next tasks so that the peak is avoided. The energy price of the next 24 hours corresponds to the considered weights of our optimization problem 2.2.

Figure 2.4 and Figure 2.5 are generated by the website energy-charts [Fra23].

2.5 Related Literature

We now consider the related literature on job-shop scheduling problems with energy prices and time windows. The job-shop scheduling problem is part of the set of classical scheduling problem variations. There are also open-shop scheduling and flow-shop scheduling problems. Furthermore, there are single-machine and parallel-machine scheduling problems. In addition, there are the traveling salesperson problem and vehicle routing problems, which are similar to scheduling problems. These problems differ in additional constraints (precedence constraints, assignment to particular machines, resource constraints). The scheduling problems also differ in the number of considered machines and the description of the jobs as a list of several tasks. Since the different scheduling problems can have some constraints in common, the valid inequalities of other scheduling problems could also be valid and should be transferred to the setting at hand.

This thesis considers the job-shop scheduling problem with energy prices and time windows. This problem is strongly related to the classical job-shop scheduling problem. If we do not consider the objective function, which in our case is the optimization of the energy costs of the resulting schedule, multiple techniques and algorithms of classical job-shop scheduling could be applied. However, unlike real-world scenarios, we assume that energy prices are known with certainty for the entire time window. Moreover, the processing and ramping durations are fixed. Therefore, the problem is deterministic, and we do not consider stochastic approaches to solve the scheduling problem. We provide a short review of job-shop scheduling-related work to embed the thesis within the context of former and modern solution approaches to classical job-shop scheduling and energy-aware scheduling. This review includes articles on problem formulations, presolving and propagation schemes, branching and separation algorithms, and techniques to determine primal solutions.

When considering the energy-aware scheduling problems in the case of integer linear programming, the resulting formulations depend on two parts: the scheduling of tasks and the computation of the energy. Therefore, the classical scheduling formulations are often reused and should be discussed before the energy-aware scheduling models are considered.

Scheduling problems arise in various forms in different research areas. A traveling salesperson problem can also be interpreted as a scheduling problem as well as a vehicle routing problem. An overview of different scheduling problems can be found in the basic books [Pin08, BJS94].

The classical job-shop scheduling problem with makespan is a well-known NP-hard combinatorial optimization problem [LR79, LK78, GJS76]. Minor changes within the objective or the problem settings can increase or decrease its computational complexity. The complexity classes of general scheduling problems in multiple settings are listed in several papers and reviews [BDP96, JM99, CPW98]. The basic theory of complexity can be read, for example, in the book of Korte and Vygen [KV12], the book of Garey and Johnson [GJ79] or the book of Arora [AB06].

Job-shop scheduling problems can be solved by integer programming or heuristically by combinatorial algorithms and metaheuristics, for example, [Pin08]. The history of computational approaches in the case of job-shop scheduling starts with the papers of Johnson [Joh53], Smith [Smi56], and Jackson [Jac57]. The authors consider different variants of job-shop scheduling with limited problem sizes and present algorithmic approaches to schedule the jobs in the optimal order to minimize the makespan or total weighted tardiness. The latter is also studied within further research [HG14, PVW85, ARPV90].

The formulation by Manne [Man60] was an early approach to solve scheduling problems by integer programming. Manne used continuous variables to describe the processing start of the jobs and binary variables to describe the precedence order of two distinct jobs. The binary variables are so-called *linear ordering variables* and are, among others, reused in the article of Grötschel et al. [GJR84, GJR85], where he considers a linear ordering problem. He also introduces additional valid inequalities and proposes a cutting-plane algorithm to solve the problem efficiently. An extension to linear ordering variables is the so-called *betweenness variables*, which were introduced in the article of Caprara [COR$^+$11] and reused in the paper of Bley, and D'Andreagiovanni and Karch [BDK13]. Betweenness variables are binary variables, fixing the order of three tasks each. Integer linear programming (ILP) models using ordering variables correspond to disjunctive programming, introduced by Balas [Bal79, Bal85]. However, disjunctive programming will have only a minor role in this thesis. Nevertheless, the disjunctive graph and the detection of valid execution orders will be used. For more insights, we refer to the book of Pinedo [Pin08] and the paper of Baptiste et al. [BLPN06].

Another technique to order the tasks on the machines is a so-called *rank-based formulation*. This formulation was introduced by Wagner [Wag59] and assigns tasks to positions within the same machine. Additional variables are introduced to describe the feasible processing start of each task from the given position on the machine.

Within this thesis, we are using a third way of formulating the scheduling problem, called *time-indexed formulation*. This way of modeling assumes that the time window can be discretized and was introduced by Bowman [Bow59]. The formulation requires a require one variable for each task and each period. The variables explicitly depict if the task starts processing in a specific period. In contrast to the previously mentioned formulations, the number of variables of this formulation depends on the size of the time window. This means that the formulation suffers from many variables when solving instances with large time windows.

Computational experiments of Ku and Beck [KB16] show that the disjunctive programming approach outperforms the time-indexed formulation regarding the objective makespan. However, the rank-based model and the disjunctive programming model offer the possibility of branching on precedences or positions within the list of jobs. The time-indexed formulation is only allowed to fix the processing of certain jobs to selected periods in each branch. Thus, problem-adapted branching rules could significantly change the results of the computational experiments. Moreover, the problem formulation could be strengthened by disaggregation of the precedence constraints. Another computational study of scheduling formulations is in the paper of Unlu [UM10]. The author analyzes the problem formulations in the case of different processing durations and claims that the time-indexed formulation is suitable if the processing durations are small in relation to the time window.

Time-indexed formulations often lead to strong dual bounds [QS94, DW90]. Additionally, the time-indexed models allow easy linking of period-dependent events and constraints to the jobs processed at that point in time, for example, the description of interruptions of links within the fiber replacing scheduling problem [BDK13]. Various discussions of time-indexed formulations of several scheduling problems with additional constraints are possible [DW90, Wol97, CS96, Art17]. Modern surveys on the different classical job-shop scheduling formulations are conducted in [XSRH22, HSRH22].

Additional constraints extend scheduling applications. There are resource-constrained scheduling problems that deal with the limited workforce employed to process the tasks [Tal82, Art13, HDD98]. Moreover, the consideration of precedence constraints often increases the computational complexity of the problem [QW91, BSV08, CRd06, MSS04]. Scheduling applications also appear with different additional constraints and need to be handled differently than without the additional constraints. There can be precedence constraints between arbitrary tasks [QW91, BSV08, CRd06, MSS04], as well as release and due dates for the jobs [DW90, SVDVZ96, BLSV98, BSV08]. Furthermore, one can consider setup times [ANCK08, LP97, Yan99] describing the parts of the tasks' processing that can be done before its predecessor running on a distinct machine is finished.

The fundamental explanations of solving integer linear programs by branch-and-bound, branch-and-cut, and branch-and-price-and-cut are described in standard works [KV12, GNM16, CCZ14, Sch86]. The corresponding algorithms and approaches will be introduced shortly in Sections 4.2, 4.3, and 4.4.

Branch-and-price uses column generation to solve the LP-relaxations, and it is introduced by Dantzig and Wolfe [DW60]. The main idea of branch-and-price is solving the branch-and-bound nodes by column generation. Column generation treats a subset of variables implicitly and only generates the variables (columns) and includes them in the problem description if needed. Column generation and some additional remarks on branch-and-price are presented in the papers [Van05, DL05, LD05, Sad19]. The Dantzig–Wolfe reformulations are also frequently used in current research, for example, in [LW18, MR23, MDL23].

Column generation was successfully applied in the case of scheduling, for instance, by van den Akker, Hurkens and Savelsbergh [AHS00, vdAvHS99]. They proposed a Dantzig–Wolfe reformulation of a time-indexed formulation. The authors present a branch-and-price-and-cut algorithm as well as a reformulation of the time-indexed single-machine scheduling formulation with weighted completion times. To that end, they replaced the original time-indexed variables with variables describing (in)complete schedules. In addition to the column generation scheme, they present the inclusion of the column generation approach into the branch-and-bound algorithm. Van den Akker derived all facet-defining inequalities of right-hand-side 1 and 2 of the non-preemptive single-machine scheduling problem within her thesis [vdA94]. The facet-defining inequalities of single-machine scheduling also apply to job-shop scheduling, as single-machine scheduling remains a subproblem. Moreover, column generation was successfully applied in scheduling-related applications, such as nurse scheduling [JSV98] or air-crew scheduling [BSSW06]. These examples visualize that a Dantzig–Wolfe reformulation is useful if the new variables group properties that need to be described by more complex formulations.

Similar valid inequalities for the time-indexed formulation of the single machine scheduling problem are derived in the publication of Sousa and Wolsey [SW92]. The authors aggregate problem constraints and derive a structure for valid cover inequalities of the generated knapsack constraints. Bergham, Spieksma and T'Kindt [BS15] extended the idea of Sousa and Wolsey to the unrelated parallel machine scheduling problem. They present new valid inequalities involving a subset of tasks on two distinct machines. Moreover, the authors suggested a presolving approach, where variables are fixed to zero by knowledge about the best incumbent and the reduced costs of the variable. The article by Applegate and Cook [AC91] presents multiple classes of valid inequalities of scheduling formulations. However, these inequalities mostly need to be considered in the case of the continuous formulation of [Man60] and are not considered within this thesis. An extensive survey of polyhedral approaches to solving scheduling problems is given in [QS94]. Therein, different problem formulations are revised, and known valid constraints and the corresponding proofs are discussed.

Scheduling problems, formulated as ILPs, can be solved by branch-and-bound. Moreover, the computation branching disjunctions should be problem-specific to guarantee efficient solution times, for example [RF81]. Van den Akker compares different branching schemes in the case of single-machine scheduling in [vdA94]. She presents variable branching (most infeasible branching), assignment constraint branching and branching based on the rank of the task. She summarizes that the different branching schemes perform quite differently and concludes that problem-adapted branching rules perform better than classical variable branching in the case of time-indexed scheduling formulations. Brucker et al. [BJS94] consider the job-shop scheduling problem with makespan. They present a branching rule, which is related to the objective of the considered scheduling problem. The branching scheme is based on searching for so-called *critical paths* within the disjunctive graph and fixing ordering decisions between tasks. The authors additionally describe strengthening approaches, like presolving, propagation and cutting planes. Caseau [CL95] describes constraint propagation for job-shop scheduling. The mentioned rules aim to reduce the variable domains and highlight the similarity of propagation and cutting planes. The developed branching algorithm supports the constraint propagation algorithm by computing the branch so that the resulting number of domain reductions will be maximized. Since the authors considered the makespan optimization, the branching rule aims to order the tasks by introducing new precedence constraints. Further branching approaches to solving job-shop scheduling problems are mentioned within the survey of Jain and Meeran [JM99].

Near-optimal primal solutions are crucial to solve integer programs efficiently by branch-and-bound. Heuristics, approximation algorithms and metaheuristics compute the primal solutions. One well-known heuristic in the field of job-shop scheduling problems is the shifting the bottleneck heuristic [ABZ88]. This heuristic solves each machine's single-machine scheduling problem one by one. Iteratively, the previously locally solved machines are reoptimized. The selected single-machine scheduling with the highest infeasibility is derived from time windows or precedence constraints. This machine is called the bottleneck. The heuristic idea can also be adapted to job-shop scheduling, minimizing total weighted tardiness [Pin08].

Additional heuristics are classical list-scheduling heuristics, which greedily add tasks to machines until all tasks are scheduled or no further tasks can be added within their time window. List scheduling is explained in standard works [Pin08, WS11]. In the case of makespan optimization, there are different ordering strategies that always provide an a priori worst-case guarantee.

Another approach is to start with an initial (feasible) solution to search the neighborhood of the current feasible solution for improvements. This procedure is called neighborhood search or tabu search, and in the case of job-shop scheduling, it is mentioned in [DT93, NS05, Yin04]. Next to algorithmic approaches, artificial intelligence solutions have become more and more popular. Genetic algorithms [GR11, SOMGSOM14, CGT96] have become more and more successful in determining (near-optimal) primal solutions. More modern approaches use deep-learning [KFH22].

After briefly mentioning the classical scheduling approaches, now, the additional consideration of energy consumption and energy-aware scheduling is considered.

Nolde and Morari [NM10] consider a scheduling problem in the application area of steel plants. The authors use a continuous time formulation, since time-indexed formulations are computationally intractable for realistic cases. However, the authors must record the events of starting and finishing the jobs. The recorded events are used to determine if the processing of a task affects a certain load interval. Thus, the resulting objective value, the energy demand and the resulting costs will be computed from continuous processing starts. Nolde and Morari state that a 10% gap solution is acceptable. The authors state that the solution process to optimality is too time-consuming. Moreover, the authors mention that idling and breakdowns of the machines need to be considered to improve the solutions, reflect reality and thus also alternative solutions.

A straightforward way to introduce energy demands and consumption into the context of a scheduling model is by introducing variables to record the machines' states during the different periods. This is done in [SOMGSOM14], where different machine states are included to describe operational states with different energy demands. In addition to the integer programming model, the author presents a genetic algorithm leading to near-optimal solutions. The author highlights using energy-aware scheduling solutions instead of makespan solutions since avoiding energy-demand peaks leads to schedules with less deployment of power generator emissions. In [LDL+14], the authors use a multi-objective scheduling method using continuous processing start variables and binary ordering variables. The first objective of this multi-objective scheduling problem is the total weighted tardiness. The second objective is the minimization of energy consumption. The authors assume constant energy demand and reduce the problem by minimizing the number of idle periods. Primal solutions are derived by a non-dominant sorting genetic algorithm. This algorithm divides the population into levels and guarantees a diversity of the population.

The proceedings [SCH+16], by Selmair, Claus, Herrmann, Bley and Trost, introduce a time-indexed formulation for the job-shop scheduling problem. The authors consider different energy prices and multiple machine states. This paper will be discussed in detail since it is the starting point of this thesis. The authors of [BŠM+18] concern a branch-and-price approach to solve a parallel machine scheduling problem with machine modes and mode-transition cost and durations. The authors present a branch-and-price and a constraint programming approach. The constraint programming approach suffers from the symmetry present in the problem setting. Moreover, they suggest an extension to accelerate the solution process. The article [MDG19] by Masmoudi et al. includes two integer programming formulations for energy-aware job-shop scheduling. The article includes a comparison of the computational performance. The authors consider a job-shop scheduling problem with additional constraints to model a power-peak limit. One ILP is a time-indexed formulation, and a second formulation is also a time-indexed formulation, extended by disjunctive variables to order the tasks and multiple variables to measure the overlap of tasks on different machines to limit the energy peak. The computational results indicate that the smaller time-indexed formulation outperforms the more complex formulation. Further integer programming models for energy-aware scheduling are presented in by Park and Ham [PH22] and by Bruzzone et al. [BAPT12]. The authors of [BAPT12] present a time-indexed flexible flow-shop formulation. The processing starts of the jobs are coupled to an energy-requirement variable. This variable is additionally bounded by a total limit. The considered objective is a combination of weighted tardiness and the makespan. The consideration of energy is considered a resource constraint. In [PH22], the authors introduce a constraint programming and an integer programming approach. The authors consider the flexible job-shop scheduling with objective makespan in combination with the total energy price. The authors conclude that shifting production to off-peak periods leads to energy savings. More approaches to describe ramping in energy-aware scheduling models are present within the articles [CGW00, ANCK08, SCH+16, SOMGSOM14]. More approaches to tackle scheduling problems concerning energy consumption or energy prices are reviewed in the surveys [WX06, GDDT16, Kou94, GHSW20, RM21].

There are various approaches for solving energy-efficient scheduling problems by genetic algorithms [MSTP15, DTG+13]. For additional information about research on algorithms to derive primal solutions for energy-aware scheduling, see [GDDT16, ZDZ+19]. There is a modern proposed algorithm for the min-cost and max-profit single machine scheduling under electricity tariffs [PR21]. The article results that the general case of this optimization problem remains NP-hard. Still, the problem can be solved using greedy algorithms and dynamic programming approaches for exceptional circumstances with identical processing times.

The scheduling approach seems to be related because the scheduling problem with relaxed integrality constraints remains on processing the tasks fractionally as cost-effectively as possible concerning the ramping and processing duration of the machines and tasks. An extensive survey of integer programming approaches to solve the unit commitment problem is in [KOW20].

A real-world application of energy-efficient scheduling is the energy consumption of computing devices. There is, for example, the possibility of increasing the speed of the machine [TV15], which influences the processing times of the jobs. Some further articles on scheduling in real-world applications are referred to in [DW19, BMAB16]. Moreover, there is also the possibility to consider social criteria. One social criterion that could be considered is the availability of workers at the machines. A further constraint is considered by Trost [TCH17] and extends the classical scheduling problem by modern aspects.

Studies have been carried out to confirm the need to include energy efficiency in the objective of scheduling manufacturing processes [BVS+11, TRM+13] as well as research on integrating energy efficiency into the modern industry [TCTB13]. The articles address the problem of cleaner production and energy usage in industry. They point out the problem of lack of energy efficiency in manufacturing processes [BVS+11, p.675]. Moreover, the problems and needs of research and change are defined to reach the goals of European improvement in energy efficiency.

Categorization of the thesis

The mentioned literature considers classical and energy-aware scheduling problems. We mentioned different ways of modeling the scheduling of the jobs and the coupling of energy consumption and processing. The different successfully applied ways of computing feasible primal solutions by combinatorial algorithms are considered, as well as branch-and-bound methods to prove optimality. Within this thesis, we use techniques from mixed-integer programming, linear programming and combinatorial optimization to accelerate the solution process of a devised branch-and-bound algorithm. To that end, we present a Dantzig–Wolfe reformulation [DW60] to model the job-shop scheduling problem with flexible energy prices and time windows. The model considers different machine states [SOMGSOM14, SCH+16]. Moreover, the problem formulation is strengthened by implicitly treating blocks of inactive periods. The problem can be solved by brand-and-price [DL05, LW18]. However, we can propose combinatorial presolving rules to reduce the number of possible variables. Among others, there are combinatorial problem-adapted versions of dominated columns and probing [ABG+20, BS15, BJS94]. Moreover, we can implement the presolving rules to be useful in propagation. Valuable inequalities from the literature are considered to strengthen the LP relaxation and to cut off (non-optimal) feasible solutions early. We analyze possible conflicts to fill the conflict graph [ANS00], implement cutting plane separation [vdAvHS99, SW92, BS15] and derive projected inequalities from linear ordering [GJR84]. The derived constraints are strengthened by lifting [Bal75]. The integer programming formulation is solved by branch-and-price and branch-and-cut. To explore the branch-and-bound tree efficiently, we develop our own branching rules, which fall into the category of constraint-branching [vdA94, FP17, Van05, RF81]. To provide a feasible primal solution, we use classical list scheduling heuristics [Pin08], neighborhood searches and dynamic programming. Moreover, we use the genetic algorithm of [GR11] to compute feasible initial solutions. Further heuristics are problem-specific implementations of classical list scheduling heuristics mentioned in [Pin08, WS11].

2.6 Complexity Analysis

Before the presentation of solution approaches to solve the job-shop scheduling problem with flexible energy prices and time windows, we first want to understand the complexity of the problem. This involves analyzing various aspects of complexity theory and their consequences. Once we have done this, we can then demonstrate how the optimal machine state assignment can be calculated for a given schedule $\mathcal{S} : O \to [T[_{\mathbb{Z}}$ using a polynomial time algorithm based on O and T. Finally, we will prove that the scheduling problem being considered is NP-hard.

Combinatorial problems are classified using computational complexity theory. This theory also investigates the relationship between different problem classes. It seeks to determine whether problems are equally difficult or if there are easier problems.

To determine if an algorithm can efficiently solve a specific problem, the problem is categorized into different complexity classes. An algorithm is assumed to be efficient if a polynomial function $f : \mathbb{N} \to \mathbb{N},\ n \mapsto n^{c_1} + c_2$, with two constants $c_1, c_2 \in \mathbb{N}$ bounds the running time while n describes the problem size. To divide the problems into efficiently solvable and not efficiently solvable ones, so-called decision problems are considered, where the solution of the problem is either yes or no. Complexity theory can be described by the usage of Turing machines, alphabets and languages. The necessary basics are explained in the books [KV12, AB06]. Since we do not require the theory of languages and alphabets, we want to reduce the definitions to the important aspects.

To define classes of problems it is common to use decision problems. For a optimization problem $\max\{c^\top x \mid x \in V\}$ the corresponding decision problem is a problem of a form

$$\text{Exist a solution } x \in V \text{ with } c^\top x \geq k$$

with $k \in \mathbb{Z}$.

Definition 2.6.1 ([Wol98] Definition 6.1). *For a problem instance X, the length of the input $L = L(X)$ is the length of the binary representation of a "standard" representation of the instance.*

Definition 2.6.2 ([Wol98] Definition 6.2). *Given a problem P, an algorithm A for the problem, and an instance X, let $f_A(X)$ be the number of elementary calculations required to run the algorithm A on the instance X. $f^*(l) = \sup\{f_A(X) : L(X) \leq l\}$ is the running time of algorithm A. An algorithm A is polynomial for a problem P if $f^*(l) = \mathcal{O}(l^p)$ for some positive integer p.*

Definition 2.6.3 ([Wol98] Definition 6.3). *NP is the class of decision problems with the property that: for any instance for which the answer is YES, there is a polynomial proof of the YES.*

Definition 2.6.4 ([Wol98] Definition 6.5). *If $P, Q \in NP$, and if each instance X of P can be converted by an algorithm into an instance Y of instance Q in $\mathcal{O}(L(X)^p)$, $p \in \mathbb{N}$ steps, then P is polynomial reducible to Q.*

Definition 2.6.5 ([Wol98] Definition 6.6). *If each problem $P \in NP$ is polynomial reducible to problem $Q \in NP$, then the problem Q is called NP-complete.*

Definition 2.6.6 ([Wol98] Definition 6.7). *An optimization problem P is NP-hard if the corresponding decision problem is NP-complete.*

In classical job-shop scheduling problems the objective makespan is used. Makespan denotes the length of time from the start to the completion of a sequence of tasks. The objective makespan describes, that the objective is the minimization of the duration of the processing.

The classical job-shop scheduling problem with objective makespan is known to be NP-hard [Pin08].

To start our analysis of the complexity of the job-shop scheduling problem with flexible energy prices and time windows, we first examine how the optimal machine states are computed for a given feasible schedule $\mathcal{S}^J : O \to [T[_\mathbb{Z}$.

Definition 2.6.7. *Let $\mathcal{S}^J : O \to [T[_\mathbb{Z}$ be a feasible schedule of an instance of the job-shop scheduling problem with flexible energy prices and time windows. The value $obj(\mathcal{S}^J)$ denotes the best objective value of valid machine states corresponding to the schedule.*

Definition 2.6.8. *Let $m \in M$ be one machine and $t_0, t_1 \in [T[_\mathbb{Z}$ with $t_0 < t_1$ two periods. The value $best(m, t_0, s_a, t_1, s_b)$ denotes the best costs of the transition from period t_0 in state $s_a \in \{off, on\}$ to period t_1 in state $s_b \in \{off, on\}$ with offline, ramping and standby states. If there is no feasible transition, the value is $obj(\mathcal{S}) = \infty$.*

The first theorem declares that $obj(\mathcal{S})$ can be efficiently computed in polynomial time using a shortest path algorithm.

Theorem 2.6.9. *Let $\mathcal{S}^J : O \to [T[_\mathbb{Z}$ be a feasible schedule of an instance of the job-shop scheduling problem with flexible energy prices and time windows. The objective $obj(\mathcal{S}^J)$ can be computed by the usage of a shortest path algorithm in $\mathcal{O}(|M|T)$.*

Proof. The proof will follow the following steps.

1. In the first place, the network $N = (D = (V, A), l)$ is created. The network represents the valid machine state switches from inactive to active, active to active and active to inactive within the time window.
2. Secondly, we show that each path from $(start)$ to (end) corresponds to a feasible machine state assignment for schedule $\mathcal{S}^J$.
3. The third part shows that the optimal machine state assignment is present within the network as a path.
4. The last part is the computation of the number of operations required for building the network, computing the shortest path and reconstructing $\mathcal{S}^M$.

The computation of the objective, resulting from the assigned machine states, can be independently done for each machine individually since the machine states are not coupled for different machines.

Therefore, let $m \in M$ be one machine. Since the schedule $\mathcal{S}^J$ is already computed, the tasks $(j,k) \in O^M_{|m}$ can be fixed to their processing starts. The machine m is blocked within certain periods, defined by the set

$$\mathcal{T} := \bigcup_{(j,k)\in O^M_{|m}} \{\mathcal{S}^J(j,k) - d^{se}_{j,k}, \ldots, \mathcal{S}^J(j,k) + d^{pr}_{j,k} - 1\}.$$

The remaining non-fixed periods $[T[_{\mathbb{Z}} \setminus \mathcal{T}$ of machine m must be used for transitions from offline to offline, offline to online, online to offline and online to online. The allowed machine states for the transitions are standby, offline, and ramping up and down. The required ramping durations must be considered and cannot be preempted.

Additional conditions are that the machine is *off* in period -1 and in period T, and the machine must be online if a task is being set up or processed in period t.

The network $N = (D = (V, A), l)$ has the following nodes and arcs.

1. The set of nodes is defined by $V = \{(on, t) \mid t \in [T[_{\mathbb{Z}}\ \} \cup \{(\mathit{off}, t) \mid t \in [T[_{\mathbb{Z}}\ \}$.
2. The set of arcs is built from the online $\rightarrow$ online arcs, the offline $\rightarrow$ offline arcs and the ramping arcs from offline to online and from online to offline. The offline $\rightarrow$ offline arcs are only present for periods $t \in [T[_{\mathbb{Z}} \setminus \mathcal{T}$. The set is defined by

$$\begin{aligned} A = &\Big\{\big((on,t),(on,t+1)\big) \mid t, t+1 \in [T[_{\mathbb{Z}}\ \Big\} \\ &\cup \Big\{\big((\mathit{off},t),(\mathit{off},t+1)\big) \mid t, t+1 \in [T[_{\mathbb{Z}} \setminus \mathcal{T}\Big\} \\ &\cup \Big\{\big((on,t),(\mathit{off},t+d^{rd}_m)\big) \mid t, t+d^{rd}_m \in [T[_{\mathbb{Z}}\ ,\ [t, t+d^{rd}_m[_{\mathbb{Z}} \cap \mathcal{T} = \emptyset\Big\} \\ &\cup \Big\{\big((\mathit{off},t),(on,t+d^{ru}_m)\big) \mid t, t+d^{ru}_m \in [T[_{\mathbb{Z}}\ ,\ [t, t+d^{ru}_m[_{\mathbb{Z}} \cap \mathcal{T} = \emptyset\Big\}. \end{aligned}$$

   The set A only contains the feasible transitions. A start of a ramp-up in period t, such that $t + d^{ru}_m - 1 \in \mathcal{T}$ is not created as well as a ramp-down in period t, such that $t \in \mathcal{T}$.
3. The arc lengths are defined as follows:

$$\begin{aligned} l_{\big((on,t),(on,t+1)\big)} &= P_t \cdot D^{st}_m && \forall\, t, t+1 \in [T[_{\mathbb{Z}} \setminus \mathcal{T} \\ l_{\big((\mathit{off},t),(\mathit{off},t+1)\big)} &= 0 && \forall\, t, t+1 \in [T[_{\mathbb{Z}} \setminus \mathcal{T} \\ l_{\big((on,t),(\mathit{off},t+d^{rd}_m)\big)} &= \sum_{q=t}^{t+d^{rd}_m-1} P_t \cdot D^{rd}_m && \forall\, t \in [T[_{\mathbb{Z}} \setminus \mathcal{T}\ [t, t+d^{rd}_m[_{\mathbb{Z}} \cap \mathcal{T} = \emptyset \\ l_{\big((\mathit{off},t),(on,t+d^{ru}_m)\big)} &= \sum_{q=t}^{t+d^{ru}_m-1} P_t \cdot D^{ru}_m && \forall\, t \in [T[_{\mathbb{Z}} \setminus \mathcal{T}\ [t, t+d^{ru}_m[_{\mathbb{Z}} \cap \mathcal{T} = \emptyset \end{aligned}$$

   and for each $(j,k) \in O^M_{|m}$ we set

$$l_{\big((on,t),(on,t+1)\big)} = P_t \cdot D^{se}_m \quad \forall\, t \in [\mathcal{S}^J(j,k) - d^{se}_{j,k}, \mathcal{S}^J(j,k)[_{\mathbb{Z}}$$

   and

$$l_{\big((on,t),(on,t+1)\big)} = P_t \cdot D^{pr}_m \quad \forall\, t \in [\mathcal{S}^J(j,k), \mathcal{S}^J(j,k) + d^{pr}_{j,k}[_{\mathbb{Z}}\ .$$

Therefore, the arc lengths $l_{u,v}$ are the energy costs of the corresponding transition of arc $(u,v) = ((s_a,t),(s_b,q))$ with $s_a, s_b \in \{on, \mathit{off}\}$ and $t, q \in [T[_{\mathbb{Z}}$.

Note that the arcs are always directed from a node $(s,t) \rightarrow (s,q)$ with $t < q$. Thus, the constructed network is acyclic. The created network can be visualized as follows: Now, we will show that each path from $(\mathit{off}, 0)$ to (off, T) in N corresponds to a feasible machine state solution.

Let P be a path from $(\mathit{off}, 0)$ to (off, T) in N. Obviously, the path visits the nodes (on, t) for $t \in \mathcal{T}$.
Suppose the path P misses at least one node (on, q) for $q \in \mathcal{T}$.

- The path cannot visit the node (off, q) since this node has either no ingoing arcs or no outgoing arcs by construction.
- The path is using a ramping arc $((\mathit{off}, t_0), (on, t_0 + d^{ru}_m))$ with $q \in [t_0, t_0 + d^{ru}_m]$. This arc cannot be used since it is forbidden by the construction of N.

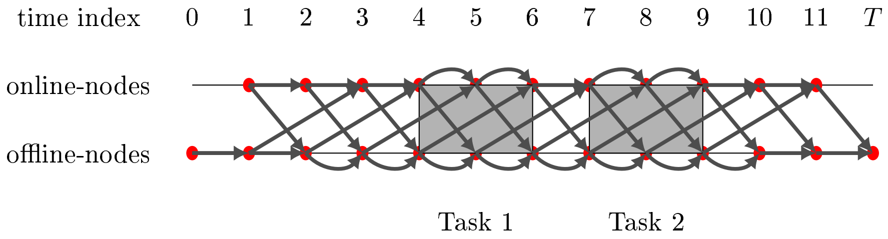


Figure 2.6: Visualization of an acyclic network to compute, for one machine with $d_m^{rd} = 1$ and $d_m^{ru} = 2$, the best matching costs for ramping and standby, if the tasks processing starts are fixed. The redundant nodes and edges are not drawn to improve the visualization.

- The path is using a ramping arc $((on, t_0), (off, t_0 + d_m^{rd}))$ with $q \in [t_0, t_0 + d_m^{rd}]$. This arc cannot be used since it is forbidden by the construction of N.

By construction, the machine needs at most d_m^{ru} periods to ramp up and d_m^{rd} periods to ramp down. Therefore, the machine cannot use shorter ramping within the path P. Thus, the representation of the ramping is valid. We only forbid ramp arcs that start or end within a period $t \in \mathcal{T}$ or that lead to the absence of a period $t \in \mathcal{T}$. As a result, the remaining ramp arcs are automatically feasible. Thus, the ramping, used by the best corresponding solution $\mathcal{S}^M$, is present within the network. Between the periods of fixed online presence of the solution $\mathcal{S}^M$, the solution can switch between online and offline if there is enough space for the ramping. Therefore, the best machine state assignment can be re-detected as a path within the network N.

The computation of the machine states from the path is straightforward. For each $m \in M$ and $t \in [T[_{\mathbb{Z}}$ set:

- $\mathcal{S}^M(m,t) = \mathit{off}$ if $((off,t),(off,t+1)$ if $t+1 \in [T[_{\mathbb{Z}}$ and $(off,t) \in V(P)$.
- $\mathcal{S}^M(m,t) = \mathit{pr}$ if $t \in [\mathcal{S}^J(j,k), \mathcal{S}^J(j,k) + d_{j,k}^{pr}[_{\mathbb{Z}}$.
- $\mathcal{S}^M(m,t) = \mathit{se}$ if $t \in [\mathcal{S}^J(j,k) - d_{j,k}^{se}, \mathcal{S}^J(j,k)[_{\mathbb{Z}}$.
- $\mathcal{S}^M(m,t) = \mathit{st}$ if $t \in [T[_{\mathbb{Z}} \setminus \mathcal{T}$ and $(on,t) \in V(P)$.
- $\mathcal{S}^M(m,q) = \mathit{ru}$ if there exists a $t \in [T[_{\mathbb{Z}}$ with $((off,t),(on,t+d_m^{ru})) \in A(P)$ and $q \in [t, t+d_m^{ru}]$.
- $\mathcal{S}^M(m,t) = \mathit{rd}$ if there exists a $t \in [T[_{\mathbb{Z}}$ with $((off,t),(on,t+d_m^{rd})) \in A(P)$ and $q \in [t, t+d_m^{rd}]$.

The network construction requires $\mathcal{O}(T)$ operations. There are at most $\mathcal{O}(T)$ nodes to denote the standby and offline periods. For each node $v \in V$, there are at most two outgoing arcs. Therefore, the number of arcs can be limited by $\mathcal{O}(T)$. Thus, the size of the network is bounded by $\mathcal{O}(T)$. Since the network is acyclic, the runtime of the shortest path algorithm is bounded by $\mathcal{O}(|V| + |A|) = \mathcal{O}(T)$. Therefore, the number of operations to compute the objective value of $\mathcal{S}^J$ is polynomially bounded in T. The number of operations required to compute $\mathcal{S}^M$ can be described as follows: for each m and t, the machine state must be computed by detecting the current period in the computed path P. This operation can be described by an iteration in a list of visited nodes to check, whether the period is assigned the start of an arc, hidden within an arc, or an offline or online node. We can limit the number of operations by T^2 per machine.

The reconstruction of the machine states requires additional $\mathcal{O}(T)$ operations to determine for each period $t \in [T[_{\mathbb{Z}}$ the corresponding machine state. The reconstruction of the machine states can be avoided by using additional labels at each node to track the machine state of the path within the algorithm.

However, the construction of the network and acyclic shortest path algorithm ensure the fast computation of the best machine states and the corresponding objective value of a given feasible schedule $\mathcal{S}$. □

Lemma 2.6.9 leads to the following lemma.

Lemma 2.6.10. *For $m \in M$, $t_0, t_1 \in [T[_{\mathbb{Z}}$ and $s_a, s_b \in \{on, off\}$, the number of operations to compute $best(m, t_0, s_a, t_1, s_b)$ is polynomially bounded in $|M|$ and T.*

We will come back to the result in sections 3, 4.1 and 4.5.

Theorem 2.6.11. *We can decide in polynomial time whether a mapping $\mathcal{S} : O \to [T[_{\mathbb{Z}}$ describes a feasible schedule.*

Proof. The feasibility of the provided schedule can be validated in polynomial time:

- The feasibility of all job-sequences can be verified in $\mathcal{O}(|O|)$, since one has to compare the start of $S(j,k)$ with a start $S(j,k+1)$ and has to verify that the difference is at least $d^{pr}_{j,k}$, for $(j,k),(j,k+1) \in O$.
- The feasibility of the schedule $\mathcal{S}$ can be validated in $\mathcal{O}(|M| \cdot T \cdot |O| \cdot T) = \mathcal{O}((|O| \cdot T)^2)$. By computing a matrix $W \in \mathbb{R}^{n_M \times T}$ with

$$W_{m,t} = \sum_{(j,k) \in O^M_{|m}} \sum_{q=t-d^{se}_{j,k}}^{t+d^{pr}_{j,k}-1} 1.0 \quad \forall\, t \in [T[_{\mathbb{Z}}$$

and validating $W_{m,t} \in \{0,1\}$ for each $m \in M$ and $t \in T$, the validity of the schedule is proven. Otherwise, two tasks are processed on the same machine, disrespecting the setup or processing times. In addition, the release and due date conditions of each task must be validated by two additional comparisons per task. Another way to verify the feasibility of the schedule is to do $\mathcal{O}(|O|^2)$ comparisons of the processing starts of the tasks on the same machine.

Thus, the verification process can be completed in $\mathcal{O}(|O|^2)$ operations. □

Theorem 2.6.12. *Given a feasible schedule $\mathcal{S}^J : O \to [T[_{\mathbb{Z}}$ one can decide in polynomial time whether a mapping $\mathcal{S}^M : M \times [T[_{\mathbb{Z}} \to \mathtt{S}$ describes a feasible machine state assignment corresponding to the schedule $\mathcal{S}^J$.*

Proof. We are given the feasible schedule $\mathcal{S}^J : O \to [T[_{\mathbb{Z}}$ and the machine state assignment $\mathcal{S}^M : M \times [T[_{\mathbb{Z}} \to \mathtt{S}$. Then, the verification of whether the machine states are set correctly is also done in polynomial time:

1. First, the correct setting of the machine states *se*, and *pr* in the period $t \in [\mathcal{S}(j,k) - d^{se}_{j,k}, \mathcal{S}(j,k) + d^{pr}_{j,k}[_{\mathbb{Z}}$ needs to be done for each task $(j,k) \in O$. We can bound them by $\mathcal{O}(|O| \cdot T)$ operations. Since for each task $(j,k) \in O$, there are at most $T \cdot 2$ queries of the machine state.
2. Secondly, if the machine states of processing and setup are without any mistakes, the further machine states must be checked. Therefore, all machines states $s \in \{st, se, pr\}$ are transformed into *on*. Then, the network of proof 2.6.9 is created. If the sequence of machine states corresponds to a $(0, \mathit{off}) - (T, \mathit{off})$ path within the network, then the machine states are set correctly, and the solution is feasible. The validation that the sequence of the machine states corresponds to a path within the created network is bounded by $\mathcal{O}(T)$ for each $m \in M$.

Thus, the verification process can be completed in $\mathcal{O}(n_M \cdot T)$ operations. □

Corollary 2.6.13. *We can decide in polynomial time, whether a schedule $\mathcal{S}^J : O \to [T[_{\mathbb{Z}}$ and a machine state assignment $\mathcal{S}^M : M \times [T[_{\mathbb{Z}} \to \mathtt{S}$ are feasible or not.*

Remark 2.6.14. *If there are additional constraints, such as the energy demand of the machines is not allowed to exceed a limit the computation of the optimal machine states can become harder, and the usage of a shortest path algorithm to compute the objective value is not guaranteed.*

Now, we show that the job-shop scheduling problem with flexible energy prices and time windows is *NP*-hard.

Theorem 2.6.15. *The job-shop scheduling problem with flexible energy prices and time windows is NP-hard.*

Proof. Let $k \in \mathbb{N}$. The decision problem for the classical job-shop scheduling problem with objective makespan, whether there exists a schedule with $C_{max} \leq k$, is *NP*-complete for $m \geq 3$ [SS95, p.239]. The mentioned decision problem can be reduced in polynomial time to the job-shop scheduling problem with flexible energy prices and time windows using the following transformation:

1. The number of machines in the job-shop scheduling problem with flexible energy prices and time windows equals the number of machines in the classical job-shop scheduling problem.
2. Jobs and tasks are copied.
3. The processing time of each task is not changed.

4. The setup time of each task is set to 0.
5. The ramping durations are set to 1 for each machine.
6. The energy demand is set to $D_m^s = 1$ for each $m \in M$ and $s \in \mathtt{S}$.
7. The time window is set to $T = 2 \cdot k$.
8. The time windows of the tasks satisfy $a_{j,k} = 0$ and $f_{j,k} = 2 \cdot T$.
9. Set $P_t = 0$ for $t \in [0, k+2[_{\mathbb{Z}}$.
10. Set $P_t = 1$ for $t \in [k+2, T[_{\mathbb{Z}}$.

Each machine must at least ramp up and down once. The required space is $d_m^{rd} + d_m^{ru} = 1 + 1 = 2$. If there exists a feasible schedule of length $\leq k$ for the classical job-shop scheduling instance, then the job-shop scheduling problem with flexible energy prices and time windows has a solution with objective 0. Suppose each solution of the job-shop scheduling problem with flexible energy prices and time windows has an objective greater than 0, but the classical job-shop scheduling problem has a solution $\mathcal{S}$ with $C_{max} \leq k$. Let $\mathcal{S}$ be the solution of the classical job-shop scheduling problem. Then, the solution $\mathcal{S}(j,k)^* = \mathcal{S}(j,k) + 1$ is a feasible schedule with $C_{max}^* \leq k+1$ and the earliest start is in period 1. The schedule $\mathcal{S}^*$ is a feasible schedule of the job-shop scheduling problem with flexible energy prices and time windows if we imply the best corresponding machine states. Since the scheduling finishes in period k, the earliest ramp-down can start in period $k+1$. Therefore, each machine is offline on period $t \geq k+2$, and the objective value is 0. This is a contradiction to the assumption.

This implies that the existence of a polynomial time algorithm for the job-shop scheduling problem with flexible energy prices and times would also solve the classical job-shop scheduling problem, which is known to be NP-hard. Thus, the job-shop scheduling problem with flexible energy prices and time windows is also NP-hard. □

Within this thesis, the execution order of the tasks is sometimes assumed to be fixed to compute feasible primal solutions. Less challenging is the analysis of the single-machine scheduling problem with the objective "flexible energy prices" and a fixed execution order of the tasks. We only provide an analysis of the single machine scheduling result.

The following theorem states that the single-machine scheduling problem with flexible energy prices and time windows can be solved in polynomial time. Therefore, we provide an algorithm to compute the solution of the single-machine scheduling problem with energy prices and a total order of the tasks and prove that the algorithm's solution time is polynomially bounded in the size of the number of tasks, the number of machines and the size of the time window.

Theorem 2.6.16. *The single-machine scheduling problem with flexible energy prices, time windows and a total order of tasks can be solved within polynomial time.*

Proof. We are given one machine and a set of n tasks $o = (j,k) \in O$. The operations $o_i \in O$ are ordered, such that $o_i \prec o_j$, if $i < j$ for $i, j \in [n[_{\mathbb{Z}}$. We build the following network $N = (D = (V, A), l)$:

- The set of nodes is defined by

$$\begin{aligned} V = &\{start, end\} \\ &\cup \{(o,t) \mid \ \forall\, o \in O,\ t \in [T[_{\mathbb{Z}}\ \}. \end{aligned}$$

- The set of arcs is defined by

$$\begin{aligned} A = \{(start, (o_0, t)) \mid t \in [T[_{\mathbb{Z}}\ \} & \\ \cup \{((o_{n-1}, t), end) \mid & \quad t \in [T[_{\mathbb{Z}}\ \} \\ \cup \{((o_a, t), (o_b, q)) \mid & \quad o_a = (j,k), o_b = (i,l) \in O, b = a+1 \\ & \quad t, q \in [T[_{\mathbb{Z}} \text{ with } q \geq t + d_{j,k}^{pr} + d_{i,l}^{se}\}. \end{aligned}$$

.

The network contains $n \cdot T + 2$ nodes and $\mathcal{O}(n \cdot T^2)$ arcs. The arc lengths are set to

$$\begin{aligned} l_{(start,(o,t)} &= best(m, 0, off, t - d_{j,k}^{se}, on) + obj(j,k,t) && \forall\, t \in [T[_{\mathbb{Z}}\ ,\ (j,k) = o \in O, \\ l_{((o,t),end)} &= best(m, t + d_{j,k}^{pr}, on, T, off) && \forall\, t \in [T[_{\mathbb{Z}}\ ,\ (j,k) = o \in O, \\ l_{((o_i,t),(o_j,q))} &= best(m, t + d_{j,k}^{pr}, on, q - d_{i,l}^{se}, on) + obj(i,l,t) && \forall\ ((o_i,t),(o_j,q)) \in A. \end{aligned}$$

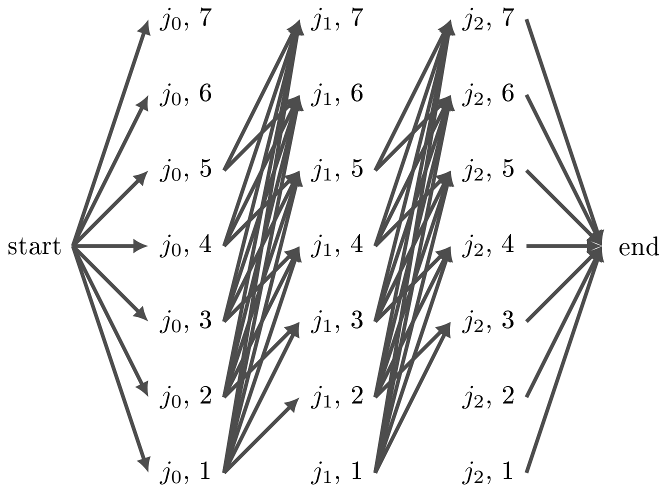


Figure 2.7: This figure visualizes the network, which is used to compute the optimal schedule with a predefined fixed execution order. In this example, we set the setup duration to 0 and the processing duration to 1 for each job. The ramping durations are set to one for ramp-up and ramp-down. Moreover, the complete time window is set to $T = 9$. There is a start node and an end node for the path. The jobs $\{j_0, j_1, j_2\}$ must be processed in the order $j_0 \prec j_1 \prec j_2$. The job j_0 can start processing in period $t \in \{1, \ldots, 7\}$. Then, the job j_1 must start processing. However, the task j_1 cannot start in period $t = 0$ since j_0 must be processed before. Therefore, the resulting digraph only contains arcs $(j_i, t) \to (j_{i+1}, q)$ with $q > t$. The arc lengths correspond to the best energy demand of the transition. Any path from *start* to *end* must define the valid processing starts of the tasks j_0, j_1 and j_2.

One example of such a graph is visualized in Figure 2.7. The computation of $obj(j, k, t)$ is polynomially bounded. The computation of $best(m, t_0, t_1)$ is polynomially bounded. This fact follows from Corollary 2.6.9. The network is acyclic. Therefore, the runtime of the shortest path algorithm is bounded by $\mathcal{O}(n \cdot T^2)$. The computation of the arc weights needs $\mathcal{O}(n \cdot T^3)$ operations. Thus, the construction and computation are polynomially bounded by the length of the time window and the number of tasks.

If at least one path exists, the shortest path within the network describes the optimal solution to the single-machine scheduling problem with flexible energy prices and a fixed total order of the tasks. Suppose we are given an optimal and feasible schedule. Then, the path through the network is fixed since the processing starts are fixed. Thus, the resulting objective value is also fixed, and the network algorithm will find the associated solution. Since the network only allows feasible processing starts and valid transitions between the start and the first processing start, between processing starts of successive tasks, and between the last processing start and end, there cannot be any further feasible solution with a lower objective than the optimal solution. Therefore, the single-machine scheduling problem with flexible energy prices can be solved within polynomial time. □

The additional consideration of start and due dates does not change the algorithm's complexity. However, the number of nodes for each task will be reduced to the set of feasible processing starts within the time windows.

In the case of multiple machines, the complexity of the job-shop scheduling problem with flexible energy prices and the total order of the tasks on the machines is unknown. However, if we allow further precedence constraints between arbitrary tasks, then the complexity of the resulting problem is *NP*-hard [Har21]. In addition, if we neglect the fixed execution order of the tasks, the problem becomes *NP*-hard if there are precedence constraints or release and due dates [LK78, LLK77].

Chapter 3

Integer Linear Programming Formulations

This chapter presents two modeling approaches for the job-shop scheduling problem with flexible energy prices and time windows. First, we review the problem formulation of [SCH$^+$16] and suggest various ways in which the model can be strengthened. The main part of this chapter is the introduction of a partial Dantzig–Wolfe reformulation of the model in [SCH$^+$16]. We only call the reformulation a partial Dantzig–Wolfe reformulation because a part of the integer solutions of the problem formulation in [SCH$^+$16] is not feasible in terms of 2.1.3 and thus, those solutions are no longer feasible in the reformulation. The partial Dantzig–Wolfe reformulation uses new variables explicitly describing intervals where the machine is inactive (ramping down, offline, and ramping up). Furthermore, the objective coefficients are assigned directly to each model variable within the reformulation. Both formulations have one thing in common: scheduling the tasks under precedence constraints. The formulations only differ in the description of the computation of the total energy price. Before introducing the problem formulations, note that the basic concepts of integer linear programming are covered particularly well in the books [KV12, WN14, CCZ14] and we do refer to these books to explain the basics.

Remark 3.0.1. *To not overload the ILP modeling with notation, the time index bounds are not specified down to the smallest detail. If the index of a summation is out of bounds, the associated sum will be assumed to start later or end earlier to run within the limits. Thus, for $a \in \mathbb{R}^T$ and $l, r \in \mathbb{Z}$ with $l \leq r$, we use the following notation:*

$$\sum_{t=l}^{r} a_i = \sum_{t=\max\{0,l\}}^{\min\{T-1,r\}} a_i.$$

3.1 A State-Based Model

The authors in [SCH$^+$16] introduced an integer programming formulation for the job-shop scheduling problem with flexible energy prices and time windows. They used time-indexed variables for the processing starts of the tasks and time-indexed variables for the machine states.

For each task $(j,k) \in O$ and for each $t \in [T[_{\mathbb{Z}}$, there exists a binary variable $x_{j,k,t} \in \{0,1\}$ with the following meaning

$$x_{j,k,t} = \begin{cases} 1, & \text{iff task } (j,k) \text{ starts processing in period } t, \\ 0, & \text{otherwise.} \end{cases}$$

The objective coefficient of the variable $x_{j,k,t}$ is zero. Additionally, as mentioned in Section 2, the setup of task (j,k) needs to be completed directly before the start of its processing. To that end, the constellation $x_{j,k,t} = 1$ means that the setup of (j,k) starts in period $t - d^{se}_{j,k}$ and the processing of task (j,k) finishes in period $t + d^{pr}_{j,k} - 1$. In period $t + d^{pr}_{j,k}$, the machine is ready to switch to another machine state.

The energy price, and thus the objective value of the schedule, is computed by linking the processing starts of the tasks to machine state variables. The machine state variable

$y^s_{m,t} \in \{0,1\}$ is created for each machine $m \in M$, each state $s \in \mathtt{S}$, and each period $t \in [T_+[_{\mathbb{Z}}$. The variable definition is as follows:

$$y^s_{m,t} = \begin{cases} 1, & \text{iff machine } m \text{ runs in period } t \text{ in state } s, \\ 0, & \text{otherwise.} \end{cases}$$

The objective coefficient of the variable $y^t_{m,s}$ is the price of the consumed energy. These costs are determined by $D^s_m \cdot P_t$ for running the machine m in state s in period t. The x-variables correspond to the schedule $\mathcal{S}^J$ and the y-variables correspond to the machine state assignment $\mathcal{S}^M$. We present the complete integer programming formulation for the job-shop scheduling problem with flexible energy prices and time windows.

$$\text{minimize} \quad \sum_{m\in M}\sum_{t\in T}\sum_{s\in\mathtt{S}} D^s_m \cdot P_t \cdot y^t_{m,s} \tag{3.1a}$$

subject to

$$\sum_{t\in[a_{j,k},f_{j,k}[_{\mathbb{Z}}} x_{j,k,t} = 1, \qquad (j,k) \in O \tag{3.1b}$$

$$\sum_{q=0}^{t-d^{pr}_{j,k}} x_{j,k,q} - \sum_{q=0}^{t} x_{j,k+1,q} \geq 0, \qquad (j,k),(j,k+1) \in O,\ t \in [T[_{\mathbb{Z}} \tag{3.1c}$$

$$\sum_{s\in\mathtt{S}} y^t_{m,s} = 1, \qquad m \in M, t \in [T_+[_{\mathbb{Z}} \tag{3.1d}$$

$$y^{-1}_{m,\mathit{off}} = 1, \qquad m \in M \tag{3.1e}$$

$$y^{T}_{m,\mathit{off}} = 1, \qquad m \in M \tag{3.1f}$$

$$\sum_{(j,k)\in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t} x_{j,k,q} \leq y^t_{m,pr}, \quad t \in [T[_{\mathbb{Z}}, m \in M \tag{3.1g}$$

$$\sum_{(j,k)\in O^M_{|m}} \sum_{q=t+1}^{t+d^{se}_{j,k}} x_{j,k,q} \leq y^t_{m,se}, \quad t \in [T[_{\mathbb{Z}}, m \in M \tag{3.1h}$$

$$\begin{aligned} &y^t_{m,pr} + y^t_{m,se} + y^t_{m,st} \\ &\quad + y^q_{m,\mathit{off}} + y^q_{m,ru} \leq 1, \qquad m \in M, t \in [T_+[_{\mathbb{Z}}, q \in [t+1, t+d^{rd}_m[_{\mathbb{Z}} \cap [T_+[_{\mathbb{Z}} \end{aligned} \tag{3.1i}$$

$$\begin{aligned} &y^t_{m,pr} + y^t_{m,se} + y^t_{m,st} \\ &\quad + y^q_{m,\mathit{off}} + y^q_{m,rd} \leq 1, \qquad m \in M, t \in [T_+[_{\mathbb{Z}}, q \in [t-d^{ru}_m, t-1[_{\mathbb{Z}} \cap [T_+[_{\mathbb{Z}} \end{aligned} \tag{3.1j}$$

$$x_{j,k,t} \in \{0,1\} \quad (j,k) \in O, t \in [T[_{\mathbb{Z}} \tag{3.1k}$$

$$y^s_{m,t} \in \{0,1\} \quad m \in M, s \in \mathtt{S}, t \in [T_+[_{\mathbb{Z}} \tag{3.1l}$$

Description of the Integer Programming Formulation

The objective (3.1a) describes the minimization of the total price of the consumed energy. The assignment constraints (3.1b) enforce that each task $(j,k) \in O$ starts processing between its earliest and latest possible start time. The inequalities (3.1c) describe the precedence constraints of successive tasks $(j,k) \in O$ belonging to the job-sequence $j \in J$. The equations (3.1e) and (3.1f) force the machine to be offline at the beginning and at the end of the time window $[T_+[_{\mathbb{Z}}$. Thus, the machine $m \in M$ must ramp up and ramp down at least once if there exist some tasks $(j,k) \in O^M_{|m}$ to process. The equations (3.1d) ensure that each machine $m \in M$ is in each period $t \in [T_+[_{\mathbb{Z}}$ in one machine state $s \in \mathtt{S}$. The constraints (3.1h) couple the machine state variable $y^{se}_{m,t}$ to the processing variables $x_{j,k,t}$ with $(j,k) \in O^M_{|m}$ and $t \in [T[_{\mathbb{Z}}$ for machine $m \in M$. The machine m must run in state *setup* in period t, if a task $(j,k) \in O$ starts processing in a period $q \in [t+1, t+d^{se}_{j,k}+1[_{\mathbb{Z}}$ with $(j,k) \in O^M_{|m}$. Analogously, the inequalities (3.1g) describe that the machine $m \in M$ needs to be in state *processing* in period $t \in [T[_{\mathbb{Z}}$, if one task $(j,k) \in O^M_{|m}$ starts processing in $q \in [t-d^{pr}_{j,k}+1, t+1[_{\mathbb{Z}}$.
The inequalities (3.1i) are the ramp-down constraints of machine m. The inequalities state that if the machine is active in period t, at least d^{rd}_m periods must pass before the machine can be in the state *offline* or *ramp-up*. Analogously, the constraints (3.1j) are the ramp-up

constraint and describe the number of necessary periods of a ramp-up of each machine $m \in M$. The inequalities describe that d_m^{ru} periods must pass if the machine is in the state *offline* or *ramp-down* in period q until the machine can be active. The integrality constraints are the remaining conditions (3.1l) and (3.1k).

The respective Polytopes

The polytope, spanned by the set of all feasible solutions to the problem formulation, is defined by

$$\mathcal{P}^S := \text{conv}(\{(x,y) \in \{0,1\}^{(O\times T + M \times \mathtt{S} \times T_+)} \mid \\ (x,y) \text{ is feasible for (3.1a)–(3.1j)}\}).$$

The polytope of the LP relaxation is defined by

$$\mathcal{P}^S_{LP} := \text{conv}(\{(x,y) \in [0,1]^{(O\times T + M \times \mathtt{S} \times T_+)} \mid \\ (x,y) \text{ is feasible for (3.1a)–(3.1j)}\}).$$

Definition 3.1.1. *The polytope*

$$\mathcal{P}^{feas} = \text{conv}(\{(x,y) \in \mathbb{N}^{n^{states}} \mid (x,y) \text{ corresponds to a feasible solution } (\mathcal{S}^J, \mathcal{S}^M)\})$$

includes all feasible solutions in terms of 2.1.3.

Our goal is to provide a problem formulation which is as close as possible to $\mathcal{P}^{feas}$ but does not exclude any of its solution. Therefore the following part presents an existing problem formulation. Afterward a new problem formulation for the job-shop scheduling problem with flexible energy prices and time windows is introduced.

Further Remarks to the Formulation

The problem formulation describes the set of all feasible schedules and the respective valid machine profiles. The ILP formulation combines a classical time-indexed scheduling formulation with a description of feasible machine state sequences. However, the ramping constraints (3.1j) and (3.1i) allow an arbitrary enlargement of the ramping if it is useful concerning the objective. Furthermore, the formulation in [SCH$^+$16] is only valid for the scenario when the energy demand satisfies $D_m^{st} < D_m^{pr}$ and $D_m^{st} < D_m^{se}$, which is a frequently occurring scenario. Moreover, the model assumes that the energy demand of machine state *standby* is smaller than the machine states *processing* or *setup*. However, running the machine on standby can be more expensive than running the machine in setup or processing. For example, heating and cooling processes to ensure operational readiness could increase the energy demand. Then, for realistic negative energy prices, the problem formulation does not necessarily compute a valid optimal solution in terms of 2.1. Therefore, here, the problem formulation gives a chance for improvement. Furthermore, one can create an integer feasible solution of (3.1a)–(3.1l) with incomplete ramping between a suitable number of offline periods, for example

$$(\ldots, \textit{rd}, \textit{rd}, \textit{rd}, \textit{off}, \textit{off}, \textit{ru}, \textit{off}, \ldots)$$

in the case of $d_m^{rd} = 3$ and $d_m^{ru} = 2$. The constraints (3.1i) and (3.1j) only ensure that the machine cannot switch directly from active to inactive. However, the computed schedule represented by x is feasible. Thus, one can compute the respective best machine states within polynomial time, see 2.6.9. Nevertheless, the model of [SCH$^+$16] contains solutions, which are infeasible in terms of the problem definition 2.1.3. The authors of [SCH$^+$16] know about the problems and only propose this formulation for instances with limited energy demand and positive energy prices. Moreover, the authors remark that there is a need for further constraints to ensure that the machines strictly satisfy the ramping durations.

3.1.1 Additional Modeling Variants

The state-based problem formulation in [SCH$^+$16], stated here as (3.1a)–(3.1k), is not used for instances with negative energy prices. The problem formulation can compute solutions where the machines can run in setup or processing, although the machines must run in standby mode to be feasible in terms of 2.1.3. This wrong assignment can be corrected by using the equations

$$\sum_{(j,k)\in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t} x_{j,k,q} = y^{pr}_{m,t} \quad t \in [T[_{\mathbb{Z}}\ , m \in M \tag{3.2}$$

$$\sum_{(j,k)\in O^M_{|m}} \sum_{q=t+1}^{t+d^{se}_{j,k}} x_{j,k,q} = y^{se}_{m,t} \quad t \in [T[_{\mathbb{Z}}\ , m \in M \tag{3.3}$$

instead of (3.1h) and (3.1g). While the inequalities (3.1h) and (3.1g) only ensure that the machine is running in state processing, respectively setup if one task is processed or set up in the given period on the machine, the constraints (3.2), respectively (3.3), ensure the machine is only allowed to run in state processing or setup if one task is processed or set up in the given period on the machine.
Therefore, if no task is set up or processed in period $t \in [T[_{\mathbb{Z}}$, but the machine should be active, the machine must idle, and the machine state computation is always consistent with the problem definition 2.2 and the description of a feasible solution 2.1.3.

Theorem 3.1.2. *The constraints* (3.2) *and* (3.3) *are valid for* $\mathcal{P}^{feas}$.

The constraints (3.2) and (3.3) describe the relationship between starting the processing of a task on the dedicated machine and assigning the correct machine state. The former constraints only describe an implication. Therefore, these constraints are valid, and the proof is not necessary.
The state-based formulation does not ensure that the machines complete at least one full ramp-up and ramp-down in fractional solutions. Moreover, the machine can run partially in the state *offline* in each period $t \in [T_+[_{\mathbb{Z}}$. The tasks are processed fractionally, since the machine is only ramped up fractionally. Then, the machines save energy by only partially ramping up and down. The following constraints can be used to improve the description of the machine's fractional ramping.

Lemma 3.1.3. *The constraints*

$$\sum_{t\in[T[_{\mathbb{Z}}} y^{ru}_{m,t} \geq d^{ru}_m \quad \forall\, m \in M : O^M_{|m} \neq \emptyset \tag{3.4}$$

and

$$\sum_{t\in[T[_{\mathbb{Z}}} y^{rd}_{m,t} \geq d^{rd}_m \quad \forall\, m \in M : O^M_{|m} \neq \emptyset \tag{3.5}$$

are valid for $\mathcal{P}^{feas}$.

The validity of the constraints is obvious. Since the machine $m \in M$ must be active to process a task $(j,k) \in O^M_{|m}$, the machine must ramp up once and ramp down once. Therefore, the number of periods with machine state *rd* or *ru* is at least given by d^{rd}_m, respectively d^{ru}_m.

Selmair et al. propose to strengthen the formulation of the ramping and machine states by adding additional constraints of similar structure as (3.1i) and (3.1j). We present further valid inequalities of the machine states.

Theorem 3.1.4. *The constraints*

$$y^{rd}_{m,t} \leq 1 - y^{off}_{m,t-1} - y^{se}_{m,t-1} \quad \forall\, m \in M,\ t \in [T_+[_{\mathbb{Z}}\ ,\ t \geq 0 \tag{3.6}$$

and

$$y^{ru}_{m,t} \leq 1 - y^{off}_{m,t+1}\ m \in M,\ t \in [T_+[_{\mathbb{Z}}\ \ t \leq T-1 \tag{3.7}$$

are valid constraints of $\mathcal{P}^{feas}$.

Proof. Let $(x,y) \in \mathcal{P}^{feas}$. Suppose the machine is in the state *rd* in period $t \in [T_+[_{\mathbb{Z}}\ ,\ t > 0$ and in the state *setup* in period $t-1$. Each task $(j,k) \in O^M_{|m}$ has a processing time of $d^{pr}_{j,k} \geq 1$ and the processing of a task must follow immediately on its setup. Since the machine runs in period t in *rd*, the processing cannot succeed the setup immediately. Thus, the machine cannot run in *rd* in period t. Therefore, the solution (x,y) is not valid in terms of 2.1.3, and the constraints (3.6) are valid for $\mathcal{P}^{feas}$.

The validity of constraints (3.7) can be proven analogously. Let $(x, y) \in \mathcal{P}^{feas}$. Suppose the machine runs in period t in *ru* and in state *off* in period $t+1$. The machine must satisfy the switching rules and cannot directly switch from *ru* to *off*. Since the ramping durations are chosen to be greater than 0, the machine cannot be in state *off* in period $t+1$. Therefore, the constraint (3.7) also describes a valid conflict. $\square$

The constraints (3.6) and (3.7) cut off the integral solutions of the form solution

$$(\ldots, y^{off}_{m,t}, y^{rd}_{m,t}, y^{off}_{m,t}, \ldots) = (\ldots, 1, 1, 1, \ldots)$$

from $\mathcal{P}^S$, which are not feasible in terms of 2.1.3.

These integer feasible solutions only appear in the cases with negative energy prices and instances, with $D^{off}_m > D^s_m$ with $s \in \{ru, rd\}$. Using the additional constraints (3.5), (3.4), (3.6), and (3.7) leads to integer solutions, without non-natural machine behavior and extensions of processing and setup. Furthermore, the point-wise ramping can be delimited. However, the number of required constraints is large, and a more compact formulation is desirable. Therefore, we exploit the fact that the variables for shutdown and startup only appear in groups of a fixed size.

Moreover, the variables for the machine states *offline*, *ramp-down*, and *ramp-up* only appear in describable sequences. This fact will be exploited within our next modeling approach.

3.2 A Partial Dantzig–Wolfe Reformulation

Before discussing the reformulation of the state-based formulation of the job-shop scheduling problem with flexible energy prices and time windows, we briefly mention some important facts concerning Dantzig–Wolfe reformulations.

3.2.1 Dantzig–Wolfe Reformulation in General

A Dantzig–Wolfe reformulation is a well-known technique to reformulate integer linear programs. A short explanation and details of its implementation are given in [DW60]. Further explanations of the basic concepts used in column generation are, for example, in [CCZ14]. We consider an integer program of the form

$$\min\{c^\top x \mid \ Ax \leq b,\ x \in \mathbb{Z}^n\}$$

with $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$ and $c \in \mathbb{Q}^n$. We partition the set of constraints into two subsystems $A_I x \leq b_I$ and $A_J x \leq b_J$ with $[m[_{\mathbb{Z}} = I \dot{\cup} J$. The set of feasible solutions of the subsystem $A_J x \leq b_J$ is described by

$$Q_J := \{x \in \mathbb{R}^n \mid \ A_J x \leq b_J,\ x \in \mathbb{Z}^n\}.$$

Then, we can rewrite the original optimization problem $\min\{c^\top x \mid \ Ax \leq b,\ x \in \mathbb{Z}^n\}$ as

$$\min\{c^\top x \mid \ A_I x \leq b_I,\ x \in \operatorname{conv}(Q_J),\ x \in \mathbb{Z}^n\}.$$

Now, let $\{v_k \in \mathbb{R}^n \mid k \in K\}$ be a finite set of all extreme points of $\operatorname{conv}(Q_J)$ and $\{r_l \in \mathbb{R}^n \mid l \in R\}$ a finite set of all extreme rays of $\operatorname{conv}(Q_J)$. Then, we can describe each point in $x \in \operatorname{conv}(Q_J)$ by

$$\begin{aligned}
& x = \sum_{k \in K} \lambda_k v_k + \sum_{l \in R} \mu_l r_r \\
& \sum_{k \in K} \lambda_k = 1 \\
& 0 \leq \lambda_k \leq 1 && \forall\, k \in K \\
& 0 \leq \mu_l && \forall\, l \in R.
\end{aligned}$$

Substituting x in the problem formulation $\min\{c^\top x \mid \; Ax \leq b, \; x \in \mathbb{Z}^n\}$ leads to the so-called Dantzig–Wolfe reformulation:

$$\begin{aligned}
\min \sum_{k\in K} \lambda_k c^\top v_k + \sum_{l \in R} \mu_l c^\top r_l \\
\text{subject to: } & \sum_{k\in K} \lambda_k A_I v_k + \sum_{l\in R} \mu_l A_I r_r \leq b_I \\
& \sum_{k \in K} \lambda_k = 1 \\
& \sum_{k\in K} \lambda_k v_k + \sum_{l \in R} \mu_l r_r \in \mathbb{Z}^n \\
& 0 \leq \lambda_k \leq 1 && \forall\, k \in K \\
& 0 \leq \mu_l && \forall\, l \in R.
\end{aligned}$$

The variables of this formulation are λ_k for $k \in K$ and μ_l for $l \in R$. The challenge is to describe $\{v_k \in \mathbb{R}^n \mid k \in K\}$ and $\{r_l \in \mathbb{R}^n \mid l \in R\}$ efficiently, since these sets can be exponentially sized. Often, those sets are too large to enumerate their members. Further details can be read in [WN14, GL10, DW60].

3.2.2 Application to Job-Shop Scheduling With Energy Prices and Time Windows

The analysis of the problem formulation (3.1a)–(3.1l) and the attempts to describe the state transitions in a meaningful way lead to the approach of introducing a Dantzig–Wolfe reformulation. One idea is the introduction of variables for each machine describing the machine states of each period $t \in [T[_{\mathbb{Z}}$. However, the number of variables is $\mathcal{O}(n_M \cdot |\mathsf{S}^T|)$ and thus too large.

We observed that the descriptions of the transitions from *offline* to *active* and from *active* to *offline* have a special property: they always appear in groups or sequences. The groups are as follows:

- Offline periods followed by d_m^{ru} ramping-up periods.
- d_m^{rd} ramping-down periods, offline periods and d_m^{u} ramping-up periods
- d_m^{rd} ramping-down periods and some offline periods.

Instead of formulating constraints to describe the ramping durations and the switching rules from inactive to active and from active to inactive, we introduce variables describing all valid intervals of inactivity. The inactive intervals always have the form: *ramping-down*, zero or more *offline* periods, and *ramping-up*. The correct ramping down and ramping up duration is always guaranteed. The main reason for this simplified notation is the implementation of the respective column generation algorithm, see Section 4.4. A further advantage of the uniform shape is the computation of the respective energy costs. The uniform shape of the inactive periods will, of course, be respected in the objective function and by an enlarged time window.

Since the uniform shape of the inactive intervals adds a ramp-down, respectively, and an additional ramp-up to some inactive intervals, the time window needs to consider that additional ramping, too. Therefore, the valid starts and ends of inactive intervals on machine $m \in M$ are chosen from the extended time window $T_B^m = [-d_m^{rd}, T + d_m^{ru}[_{\mathbb{Z}}$ to encode the constraints (3.1e), (3.1f) by inactive intervals of the similar form. Otherwise, we would describe that the machine has to initially ramp down before the machine can be ramped up for processing the tasks and shorten the time window similarly. Note that T_B^m contains periods, which are irrelevant within the following context. Note that T_B^m is a set, while T is an integer.

Now, we can introduce our structure, describing the inactive intervals on the machines.

Definition 3.2.1 (Definition of a break)**.** *Let $m \in M$ be one machine. The tuple $(t_0, t_1) \in T_B \times T_B$ with $t_1 - t_0 \geq d_m^{rd} + d_m^{rd}$ and $t_1 > t_0$ is called a* ***break*** *on machine $m \in M$.*

The name *break* is motivated by the association of being inactive by leaving the workstation, resting and going back to the workstation.

A break $(t_0, t_1) \in B_m$ satisfying $t_0 = -d_m^{rd}$ is called an initial break, and the ramp-down from t_0 to 0 is called the initial ramp-down. A break $(t_0, t_1) \in B_m$ satisfying $t_1 = T + d_m^{ru}$ is called a final break, and the ramp-up from T to $T + d_m^{ru}$ is called the final ramp-up. An example of a break (t_0, t_1) is visualized in Figure 3.1.

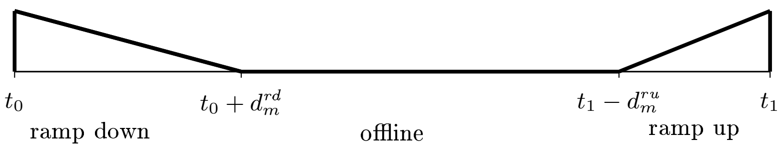


Figure 3.1: Visualization of a break and the respective machine states.

For each break $(t_0, t_1) \in B_m$, we create a binary variable $z^{rd,ru}_{m,t_0,t_1}$ to describe whether the machine is inactive in the interval $[t_0, t_1[_{\mathbb{Z}}$. Moreover, the respective machine states are treated implicitly: the machine m is starting to *ramp-down* at t_0, is in state *offline* from $t_0 + d_m^{rd}$ until inclusive period $t_1 - d_m^{ru} - 1$, and in *ramp-up* from $t_1 - d_m^{ru}$ to $t_1 - 1$. The machine can be active in period t_1, if $t_1 \neq T + d_m^{ru}$ holds.

The set of all breaks belonging to a specific machine is necessary to describe all valid variables.

Definition 3.2.2 (Set of all breaks)**.** *Let $m \in M$ be one machine. The set*

$$B_m := \{(t_0, t_1) \in T_B \times T_B \mid \ t_1 - t_0 \geq d_m^{ru} + d_m^{rd}\}$$

is the set of all breaks belonging to machine m.

Note that B_m contains breaks that cannot be used in a feasible integral solution in terms of problem definition 2.2. These variables will be detected and deleted in a presolving step, see Chapter 4.1.

The usage within the beginning and the end of the time window of the break variables is visualized in Figure 3.2.

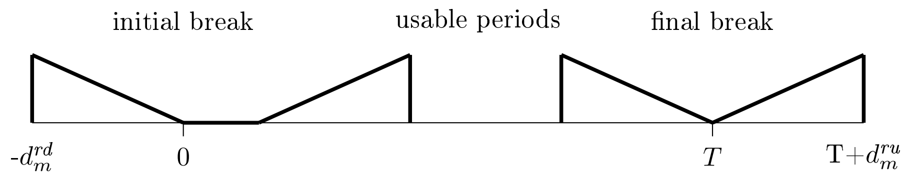


Figure 3.2: Illustration of the usage of the expanded time window.

We set the energy price $P_t = 0$ for each $t \in [T_B^m[_{\mathbb{Z}} \setminus [T[_{\mathbb{Z}}$ to ensure that the objective value of a feasible solution is not affected by the design of the breaks. These energy prices do not affect further variables except the breaks.

The objective coefficient of the break variable $z^{rd,ru}_{m,t_0,t_1}$, for $m \in M$ and $(t_0, t_1) \in B_m$ is

$$\hat{d}_{t_0,t_1,m} := \sum_{q=t_0}^{t_0+d_m^{rd}-1} P_q D_m^{rd} + \sum_{q=t_1-d_m^{ru}}^{t_1-1} P_q D_m^{ru} \ . \tag{3.8}$$

The objective coefficient $\hat{d}_{t_0,t_1,m}$ depicts the energy price of ramping the machine down within the periods t_0 to $t_0 + d_m^{rd} - 1$ and ramping the machine up between the periods $t_1 - d_m^{ru}$ to $t_1 - 1$. Thus, the usage of the variables $y^{rd}_{m,t}$, $y^{off}_{m,t}$, $y^{ru}_{m,t}$ becomes redundant for each $t \in [T[_{\mathbb{Z}}$ and for each $m \in M$, since the objective costs and the assignment of the periods to the respective machine states are well defined by the usage of the break. Since the energy price for the initial ramp-down and the final ramp-up are 0, the objective is set correctly for initial and final breaks.

The machine state variables for state *setup* and state *processing* are redundant since the energy costs can be directly linked to the processing start variables. Moreover, our reformulation does not require the explicit machine state variables to formulate the ramping constraints (3.1i) and (3.1j).

Thus, the machine state of machine $m \in M$ in period $t \in [t - d^{se}_{j,k}, t + d^{pr}_{j,k}[_{\mathbb{Z}}$ is fixed, if the task $(j,k) \in O^M_{|m}$ starts processing in period $t \in [a_{j,k}, f_{j,k}[_{\mathbb{Z}}$. Therefore, the respective machine state variables are redundant since we can directly link the respective energy costs to the processing start variable. Furthermore, the machine state variables are not necessary anymore to formulate the transition constraints. Thus, the objective coefficient of starting the processing of task $(j,k) \in O$ in period $t \in [T[_{\mathbb{Z}}$ is

$$\hat{c}_{j,k,t} = \sum_{q=t-d^{se}_{j,k}}^{t-1} P_q D^{se}_{m_{j,k}} + \sum_{q=t}^{t+d^{pr}_{j,k}-1} P_q D^{pr}_{m_{j,k}}. \tag{3.9}$$

The remaining machine state is *standby*. We introduce a variable $z^{st}_{m,t}$ for each $m \in M$ and $t \in [T[_{\mathbb{Z}}$ to describe the standby usage of machine m in period t. The variable corresponds to $y^t_{m,st}$. However, the variables describing the machine states are removed. Thus, we changed the variable name for standby usage.

Now, we list the used variables and their formal description and definition. For each $(j,k) \in O$ and each period $t \in [T[_{\mathbb{Z}}$, there is the **task variable**

$$x_{j,k,t} = \begin{cases} 1, & \text{iff task } (j,k) \text{ starts processing in period } t, \\ 0, & \text{otherwise.} \end{cases}$$

For each $m \in M$ and $t \in [T[_{\mathbb{Z}}$, there is the **standby variable**

$$z^{st}_{m,t} = \begin{cases} 1, & \text{iff machine } m \text{ is in state standby in period } t, \\ 0, & \text{otherwise.} \end{cases}$$

There is a **break variable** for each $m \in M$ and $(t_0, t_1) \in B_m$, denoted by

$$z^{rd,ru}_{m,t_0,t_1} = \begin{cases} 1, & \text{iff machine } m \text{ performs a break from } t_0 \text{ to } t_1, \\ 0, & \text{otherwise.} \end{cases}$$

The following integer linear program describes the feasible solutions of the job-shop scheduling problem with flexible energy prices and time windows.

$$\text{minimize} \quad \sum_{m \in M} \Big(\sum_{t \in [T[_{\mathbb{Z}}} \big(P_t D^{st}_m z^{st}_{m,t} + \sum_{(j,k) \in O^M_{|m}} \hat{c}_{j,k,t} x_{j,k,t} \big) + \sum_{(t_0,t_1) \in B_m} \hat{d}_{t_0,t_1,m} z^{rd,ru}_{m,t_0,t_1} \Big) \tag{3.10a}$$

subject to

$$\sum_{t \in [a_{j,k}, f_{j,k}[_{\mathbb{Z}}} x_{j,k,t} = 1, \qquad (j,k) \in O \tag{3.10b}$$

$$\sum_{q=0}^{t-d^{pr}_{j,k}} x_{j,k,q} - \sum_{q=0}^{t} x_{j,k+1,q} \geq 0, \qquad t \in [T[_{\mathbb{Z}},\ (j,k),(j,k+1) \in O \tag{3.10c}$$

$$\sum_{(j,k) \in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t+d^{se}_{j,k}} x_{j,k,q} + z^{st}_{m,t} + \sum_{\substack{(t_0,t_1) \in B_m: \\ t \in [t_0,t_1[_{\mathbb{Z}}}} z^{rd,ru}_{m,t_0,t_1} = 1, \qquad m \in M,\ t \in [T[_{\mathbb{Z}} \tag{3.10d}$$

$$\sum_{(-d^{rd}_m, t_1) \in B_m} z^{rd,ru}_{m,-d^{rd}_m,t_1} = 1, \qquad m \in M \tag{3.10e}$$

$$\sum_{(t_0, T+d^{ru}_m) \in B_m} z^{rd,ru}_{m,t_0,T+d^{ru}_m-1} = 1, \qquad m \in M \tag{3.10f}$$

$$x_{j,k,t} \in \{0,1\}, \qquad (j,k) \in O,\ t \in [T[_{\mathbb{Z}} \tag{3.10g}$$

$$z^{rd,ru}_{m,t_0,t_1} \in \{0,1\}, \qquad m \in M, (t_0,t_1) \in B_m \tag{3.10h}$$

$$z^{st}_{m,t} \in \{0,1\}, \qquad m \in M, t \in [T[_{\mathbb{Z}} \tag{3.10i}$$

Problem Description

The objective (3.10a) describes the total energy cost generated by the processing and setting up of the tasks, running on the machines, and energy costs by standby and ramping the machines up and down. The equations (3.10b) ensure that each task is started and processed once within its release and due date. The constraints (3.10c) describe the precedence relations between consecutive tasks of each job. The equations (3.10d) enforce that each machine is either processing or setting up a task or using a break or running in standby in each period of the original time window $[T[_{\mathbb{Z}}$. The equations (3.10e) and (3.10f) ensure each machine is inactive in the periods 0 and T by fixing the machine to use the corresponding breaks. The remaining constraints (3.10g), (3.10h), and (3.10i) are the integrality conditions of the defined variables.

Dantzig–Wolfe reformulation is often brought into connection with the column generation technique, since a Dantzig–Wolfe reformulation can suffer from an exponential sized set of variables. In Section 4.4 a column generation approach considering the break-variables is introduced.

The Polytopes

The dimension of the solution space is $n_{break} = |O \times [T[_{\mathbb{Z}}| + |M \times [T[_{\mathbb{Z}}| + \sum_{m\in M} |B_m|$, and the polytope is defined by

$$\begin{aligned}\mathcal{P}^B := \operatorname{conv}(\{(x, z^{st}, z^{rd,ru}) \in \; & \{0,1\}^{n_{break}} \mid \\ & (x, z^{st}, z^{rd,ru}) \text{ is valid for (3.10b)–(3.10i)}\})\end{aligned} \tag{3.11}$$

and the polytope of the LP-relaxation

$$\begin{aligned}\mathcal{P}^B_{LP} := \operatorname{conv}(\{(x, z^{st}, z^{rd,ru}) \in \; & [0,1]^{n_{break}} \mid \\ & (x, z^{st}, z^{rd,ru}) \text{ is valid for (3.10b)–(3.10d)}\}).\end{aligned} \tag{3.12}$$

3.2.3 Special Properties of the Polytopes

The state-based formulation (3.1a)–(3.1l) includes a classical scheduling formulation and a linkage to the description of the valid machine state transitions. The valid machine state switches are only described by conflicts disallowing invalid switches. The machine state variables are used to compute the respective objective value. The reformulation (3.10b)–(3.10i) is a classical scheduling model, extended by the constraints (3.10d), (3.10e), (3.10f) forcing to declare each period either to be blocked by a task, standby or a break.

Due to the encoding of the breaks, the problem description has

$$\mathcal{O}(|O| \cdot T + n_M T + T^2 \cdot n_M) = \mathcal{O}(T \cdot n_M \cdot T + n_M T + T^2 \cdot n_M) = \mathcal{O}(T^2 \cdot n_M)$$

binary variables. One important property of the break-based formulation is that we can relax the integrality conditions (3.10h) and (3.10i). Thus, the break and the standby variables could be chosen as continuous variables and become automatically integral if the task variables are integral. Now, we need the following well-known theorems and definitions, which are also present in standard works, for example, [KV12, p. 125 -129].

Definition 3.2.3 (Totally unimodular matrix)**.** *Let $A \in \mathbb{Z}^{m\times n}$ be a matrix with $n, m \in \mathbb{N}$. The matrix A is totally unimodular if the determinant of each quadratic submatrix of A is in $\{0, 1, -1\}$.*

Definition 3.2.4. *Let $P \subseteq \mathbb{R}^n$ be a rational polyhedron with $n \in \mathbb{N}$. We call P an integral polyhedron, if $P = \operatorname{conv}(P \cap \mathbb{Z}^n)$.*

Theorem 3.2.5. *Let $A \in \mathbb{Z}^{m\times n}$ be totally unimodular matrix. Then, for all integral $b \in \mathbb{Z}^m$, the polyhedron $P = \{x \mid Ax = b,\ x \geq 0\}$ is an integral polyhedron.*

Theorem 3.2.6. *Let $A \in \mathbb{Z}^{m\times n}$ be a totally unimodular matrix. Then, for all $b \in \mathbb{Z}^m$ the nonempty polyhedron $\{x \in \mathbb{R}^n \mid Ax = b,\ x \geq 0\}$ has an integral point in each minimal face.*

Using those theorems, we can prove that the break and standby variables can be continuous, and the integrality of the task variables directly leads to integral values for standby and break variables.

Theorem 3.2.7. *Let $(x^*, z^{st*}, z^{rd,ru*}) \in \mathcal{P}^B$ be an integer feasible solution. Then, the polytope ${\mathcal{P}^B_{LP}}^{fixed} = \{(x, z^{st}, z^{rd,ru}) \in \mathcal{P}^B \mid \ x_{j,k,t} = {x_{j,k,t}}^* \ \forall\ ((j,k),t) \in O \times [T[_{\mathbb{Z}}\}$ is integral.*

Proof. The idea of the proof is as follows: We take a feasible integral solution of the task variables. Then, we analyze the problem formulation after fixing the feasible integer solution values of the x-variables and show that the remaining inequalities and variables build a totally unimodular matrix.
Let $(x^*, z^{st*}, z^{rd,ru*}) \in \mathcal{P}^B$ a feasible integral solution. Now, we fix the task variables to the values of the integral solution. The resulting polytope can be described by the following formulation:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{m\in M} \Big(\sum_{t\in[T[_{\mathbb{Z}}} C_t D^{st}_m z^{st}_{m,t} + \sum_{(t_0,t_1)\in B_m} \hat{d}_{t_0,t_1,m} z^{rd,ru}_{m,t_0,t_1} \Big) \\
\text{subject to} \quad & \\
& \sum_{(-d^{rd}_m,t_1)\in B_m} z^{rd,ru}_{m,-d^{rd}_m,t_1} = 1, && m \in M \\
& z^{st}_{m,t} + \sum_{\substack{(t_0,t_1)\in B_m:\\ t\in\{t_0,\dots,t_1\}}} z^{rd,ru}_{m,t_0,t_1} = 1 - \sum_{(j,k)\in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t+d^{se}_{j,k}} x_{j,k,q}{}^*, && m \in M,\ t \in [T[_{\mathbb{Z}} \\
& \sum_{(t_0,T+d^{ru}_m-1)\in B_m} z^{rd,ru}_{m,t_0,T+d^{ru}_m-1} = 1, && m \in M \\
& z^{rd,ru}_{m,t_0,t_1} \geq 0, && m \in M, (t_0,t_1) \in B_m \\
& z^{st}_{m,t} \geq 0, && m \in M,\ t \in [T[_{\mathbb{Z}} \ .
\end{aligned}
$$

Let A be the associated coefficient matrix of this optimization problem. Then, A has only entries in $\{0, 1\}$. The right-hand-side b has also only entries in $\{0, 1\}$.

Each break variable $z^{rd,ru}_{m,t_0,t_1}$ appears only within the constraint (3.10d), (3.10e), (3.10f), from period t_0 to period $t_1 - 1$ on machine m. The standby variable $z^{st}_{m,t}$ only appears within the t-th (3.10d) constraints of machine m. The remaining entries of those columns are zero. If the model is built for each machine as a block, sorted by machine index and time period, then the consecutive one's matrix property is visible.

The matrix A is a consecutive one's matrix known to be totally unimodular. Since the right-hand-side b integral, the polytope of the discussed optimization problem is integral. Therefore, at least one vertex exists with integral break and standby variables respective to the fixed x. □

Theorem 3.2.8 can be used to provide a second proof of Theorem 2.6.9. We have already shown that we can compute the objective value of a schedule in polynomial time.

Within the analysis of the problem formulation (3.1b)– (3.1l), we mention the existence of integral solutions where the ramping duration is enlarged. However, the ramping duration is fixed and should not be enlarged to reduce the objective value. This enlargement is not possible in the case of the problem formulation using break variables. Thus, the problem formulation (3.1b)– (3.1l) also contains (infeasible) integral solutions that cannot be created by the reformulation (3.10b)–(3.10i).

Although there is the presumption that the reformulation is a more precise description of the convex hull of the feasible solutions in terms of 2.2 than the state-based formulation, this is to be proven. To compare the polytopes, we project the solution $(x, z^{st}, z^{rd,ru})$ into the space of the variables (x, y). We define a suitable transformation of the solution values of the breaks to the machine state variables for *offline*, *ramp-up* and *ramp-down*.

Theorem 3.2.8. *Consider the polytopes $\mathcal{P}^B_{LP}$ and $\mathcal{P}^S_{LP}$. Then, the relation $\mathcal{P}^B_{LP} \subseteq \mathcal{P}^S_{LP}$ holds within the linear space $\mathbb{R}^{n_{states}}$, with $n_{states} = |O| \cdot T + n_M \cdot |\mathsf{S}| \cdot |T_+|$.*

Proof. Let $(x, z^{st}, z^{rd,ru}) \in \mathcal{P}^B_{LP}$ a feasible (fractional) solution of the LP relaxation of the break-based formulation. The following linear transformation projects the solution $(x, z^{st}, z^{rd,ru})$ into the space of the variables (x, y). The projection

$$\Psi : \mathbb{R}^{n_{breaks}} \to \mathbb{R}^{n_{states}},\ (x, z^{st}, z^{rd,ru}) \mapsto (x, y)$$

maps the breaks and standby variables to the machine state variables y. The mapping is defined as follows:

$$
\begin{aligned}
y^{off}_{m,t} &= \sum_{\substack{(t_0,t_1)\in B_m:\\ t\in[t_0+d^{rd}_m,t_1-d^{ru}_m-1[_{\mathbb{Z}}}} z^{rd,ru}_{m,t_0,t_1}\\
y^{ru}_{m,t} &= \sum_{\substack{(t_0,t_1)\in B_m:\\ t\in[t_1-d^{ru}_m,t_1[_{\mathbb{Z}}}} z^{rd,ru}_{m,t_0,t_1}\\
y^{se}_{m,t} &= \sum_{(j,k)\in O^M_{|m}} \sum_{q=t+1}^{t+d^{se}_{j,k}} x_{j,k,q}\\
y^{pr}_{m,t} &= \sum_{(j,k)\in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t} x_{j,k,q}\\
y^{st}_{m,t} &= z^{st}_{m,t}\\
y^{rd}_{m,t} &= \sum_{\substack{(t_0,t_1)\in B_m:\\ t\in[t_0,t_0+d^{rd}_m[_{\mathbb{Z}}}} z^{rd,ru}_{m,t_0,t_1}
\end{aligned}
$$

and for each $m \in M$ and $t \in [T_+[_{\mathbb{Z}} \setminus [T[_{\mathbb{Z}}$

$$
y^{off}_{m,t} = \sum_{\substack{(t_0,t_1)\in B_m:\\ t\in[t_0,t_1[_{\mathbb{Z}}}} z^{rd,ru}_{m,t_0,t_1}.
$$

The proposed linear transformation of the variables $(x, z^{rd,ru}, z^{st})$ to the solution (x, y) need to be checked for feasibility of $\mathcal{P}^S_{LP}$ in the space of the (x, y) variables

- The task variables satisfy the constraints (3.1b), which are part of the formulation (3.10b)–(3.10i), namely constraint (3.10b).
- The task variables satisfy the constraints (3.1c), which are also part of the formulation (3.10b)–(3.10i), namely (3.10c).
- The task variables satisfy the constraints (3.1h), since for each $m \in M$ and $t \in [T[_{\mathbb{Z}}$, the inequality

$$
\sum_{(j,k)\in O^M_{|m}} \sum_{q=t+1}^{t+d^{se}_{j,k}} x_{j,k,q} \le y^{se}_{m,t} = \sum_{(j,k)\in O^M_{|m}} \sum_{q=t+1}^{t+d^{se}_{j,k}} x_{j,k,q}
$$

  holds.
- Analogously the task variables satisfy the constraints (3.1g), since each $m \in M$ and $t \in [T[_{\mathbb{Z}}$, the inequality

$$
\sum_{(j,k)\in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t} x_{j,k,q} \le y^{pr}_{m,t} = \sum_{(j,k)\in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t} x_{j,k,q}
$$

  holds.
- The machine state variables satisfy the constraints (3.1e) and (3.1f) because of

$$
y^{off}_{m,t} = \sum_{\substack{(t_0,t_1)\in B_m:\\ t\in[t_0,t_1[_{\mathbb{Z}}}} z^{rd,ru}_{m,t_0,t_1} \forall\ m \in M\ , t \in [T_+[_{\mathbb{Z}} \setminus [T[_{\mathbb{Z}}
$$

  and constraints (3.10e) and (3.10f).
- The machine state variables satisfy the ramping constraints (3.1i). Let $m \in M$, $t \in [T[_{\mathbb{Z}}$ and $q \in [t+1, t+d^{rd}_m+1[_{\mathbb{Z}}$, then substituting the y within the inequality

$$
y^q_{m,off} + y^q_{m,ru} + y^t_{m,pr} + y^t_{m,se} + y^t_{m,st} \le 1
$$

by their linear transformation's counterpart leads to

$$\begin{aligned}
& y^q_{m,off} + y^q_{m,ru} + y^t_{m,pr} + y^t_{m,se} + y^t_{m,st} && = \\
& y^q_{m,off} + y^q_{m,ru} + z^{st}_{m,t} + \sum_{(j,k)\in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t+d^{se}_{j,k}} x_{j,k,q} && = \\
& \sum_{\substack{(t_0,t_1)\in B_m:\\ q\in[t_0+d^{rd}_m,t_1[_{\mathbb{Z}}}} z^{rd,ru}_{m,t_0,t_1} + z^{st}_{m,t} + \sum_{(j,k)\in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t+d^{se}_{j,k}} x_{j,k,q} && \leq \\
& \sum_{\substack{(t_0,t_1)\in B_m:\\ q\in[t_0,t_1[_{\mathbb{Z}}}} z^{rd,ru}_{m,t_0,t_1} + z^{st}_{m,t} + \sum_{(j,k)\in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t+d^{se}_{j,k}} x_{j,k,q} && =1.
\end{aligned}$$

The last inequality holds, since

$$\sum_{\substack{(t_0,t_1)\in B_m:\\ q\in[t_0,t_1[_{\mathbb{Z}}}} z^{rd,ru}_{m,t_0,t_1} \geq \sum_{\substack{(t_0,t_1)\in B_m:\\ q\in[t_0+d^{rd}_m,t_1[_{\mathbb{Z}}}} z^{rd,ru}_{m,t_0,t_1}$$

holds. Thus, the ramping-down constraints (3.1i) is fulfilled by y.

- Analogously, for each $m \in M$, $t \in [T[_{\mathbb{Z}}$ and $q \in [t - d^{ru}_m, t-1[_{\mathbb{Z}}$, the transformed y lead to

$$\begin{aligned}
& y^t_{m,pr} + y^t_{m,se} + y^t_{m,st} + y^q_{m,off} + y^q_{m,rd} && = \\
& \sum_{\substack{(t_0,t_1)\in B_m:\\ q\in[t_0,t_1-d^{ru}_m[_{\mathbb{Z}}}} z^{rd,ru}_{m,t_0,t_1} + z^{st}_{m,t} + \sum_{(j,k)\in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t+d^{se}_{j,k}} x_{j,k,q} && \leq \\
& \sum_{\substack{(t_0,t_1)\in B_m:\\ q\in[t_0,t_1[_{\mathbb{Z}}}} z^{rd,ru}_{m,t_0,t_1} + z^{st}_{m,t} + \sum_{(j,k)\in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t+d^{se}_{j,k}} x_{j,k,q} && =1.
\end{aligned}$$

Thus, the projected solution $\psi(x, z^{st}, z^{rd,ru})$ is a feasible solution of $\mathcal{P}^S_{LP}$. □

We already have stated that $\mathcal{P}^S \neq \psi(\mathcal{P}^B)$ in $\mathbb{R}^{n_{states}}$ holds because there are integral solutions of $\mathcal{P}^S$ that cannot be linearly transformed to feasible solutions of $\mathcal{P}^B$ in $\mathbb{R}^{n_{states}}$. We provide an example of non-negative energy prices and show an example of a fractional solution of $\mathcal{P}^S_{LP}$ that cannot be represented in $\mathcal{P}^B_{LP}$ within the space $\mathbb{R}^{n_{break}}$. This will support the focus of concentrating on the break-based formulation.

Remark 3.2.9. *Consider the polytope $\mathcal{P}^S_{LP}$ and the projection $\psi(\mathcal{P}^B_{LP})$ onto the space $\mathbb{R}^{n_{states}}$ of the (x, y). Then, there exists an instance satisfying the relation $\psi(\mathcal{P}^B_{LP}) \subsetneq \mathcal{P}^S_{LP}$.*

Consider the following instance. The time window is set to $[T[_{\mathbb{Z}} = \{0, \ldots, 50\}$ and the machine and job data is defined as follows in Table 3.1

Table 3.1: Data of the instance.

j	k	$m_{j,k}$	$d^{pr}_{j,k}$	$d^{se}_{j,k}$
0	0	0	3	4
0	1	1	4	5
0	2	2	3	4
1	0	2	4	5
1	1	1	3	4
1	2	0	3	4
2	0	2	3	4
2	1	0	3	4
2	2	1	4	5

id	$m=1$	$m=2$	$m=3$
d^{ru}_m	3	3	3
d^{rd}_m	10	10	10
D^{off}_m	0	0	0
D^{ru}_m	1	1	1
D^{se}_m	2	2	2
D^{pr}_m	2	2	2
D^{st}_m	1	1	1
D^{rd}_m	10	10	10

Each job has the release date 0 and the due date $T = 50$. The energy price is $P_t = 1$ for each $t \in [T[_{\mathbb{Z}} \setminus \{20\}$. For $t = 20$, the energy price is set to $P_{20} = 0$. One can easily validate that the constraints (3.1i) and (3.1j) hold. In the period of $t = 20$, the ramp-down starts without the machine being ramped up completely. The problem formulation $\mathcal{P}^B_{LP}$ forces to block d^{rd}_m successive periods if the machine needs to be ramped down in one period. Thus, the fractional solution cannot be feasible for $\mathcal{P}_{LP}^{B}$ within the space of the $(x, z^{st}, z^{rd,ru}, y)$-variables.

Table 3.2: Excerpt from a solution of the LP-relaxation of the state variable formulation.

m	t	price	$y_{m,t}^{off}$	$y_{m,t}^{ru}$	$y_{m,t}^{se}$	$y_{m,t}^{pr}$	$y_{m,t}^{st}$	$y_{m,t}^{rd}$
2	11	1	$\frac{1}{3}$	$\frac{2}{3}$	0	0	0	0
2	12	1	$\frac{1}{3}$	$\frac{2}{3}$	0	0	0	0
2	13	1	$\frac{1}{3}$	$\frac{2}{3}$	0	0	0	0
2	14	1	$\frac{1}{3}$	0	$\frac{2}{3}$	0	0	0
2	15	1	$\frac{1}{3}$	0	$\frac{2}{3}$	0	0	0
2	16	1	$\frac{1}{3}$	0	$\frac{2}{3}$	0	0	0
2	17	1	$\frac{1}{3}$	0	$\frac{1}{3}$	$\frac{1}{3}$	0	0
2	18	1	$\frac{1}{3}$	0	0	$\frac{2}{3}$	0	0
2	19	1	$\frac{1}{3}$	0	0	$\frac{2}{3}$	0	0
2	20	1	0	0	0	$\frac{2}{3}$	0	$\frac{1}{3}$
2	21	1	$\frac{1}{3}$	0	$\frac{1}{3}$	$\frac{1}{3}$	0	0
2	22	1	$\frac{1}{3}$	0	$\frac{1}{3}$	$\frac{1}{3}$	0	0
2	23	1	$\frac{1}{3}$	0	$\frac{2}{3}$	0	0	0
2	24	1	$\frac{1}{3}$	0	$\frac{2}{5}$	$\frac{4}{15}$	0	0
2	25	1	$\frac{1}{3}$	0	$\frac{1}{3}$	$\frac{1}{3}$	0	0
2	26	1	$\frac{1}{3}$	0	$\frac{1}{3}$	$\frac{1}{3}$	0	0
2	27	1	$\frac{1}{3}$	0	0	$\frac{2}{3}$	0	0
2	28	1	$\frac{1}{3}$	0	$\frac{4}{15}$	$\frac{2}{5}$	0	0
2	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
2	61	1	1	0	0	0	0	0

The solution presented in Table 3.2 is feasible. We only verify the feasibility for a selected number of periods. The verification of the machine state transition constraints (3.1i) looks as follows:

$$\begin{aligned}
&(t,q) && y_{m,off}^{q} + y_{m,ru}^{q} + y_{m,pr}^{t} + y_{m,se}^{t} + y_{m,st}^{t} \leq 1 \\
&(t=18, q=19) && \frac{1}{3} + 0 + \frac{2}{3} + 0 + 0 = 1 \leq 1 \\
&(t=18, q=20) && 0 + 0 + \frac{2}{3} + 0 + 0 = \frac{2}{3} \leq 1 \\
&(t=18, q=21) && \frac{1}{3} + 0 + \frac{2}{3} + 0 = 1 \leq 1 \\
&\vdots && \vdots
\end{aligned}$$

The verification of the machine state transition constraints (3.1j) looks as follows:

$$\begin{aligned}
&(t,q) && y_{m,pr}^{t} + y_{m,se}^{t} + y_{m,st}^{t} + y_{m,off}^{q} + y_{m,rd}^{q} \leq 1 \\
&(t=22, q=21) && \frac{1}{3} + 0 + \frac{1}{3} + \frac{1}{3} + 0 = 1 \leq 1 \\
&(t=22, q=20) && 0 + \frac{1}{3} + \frac{1}{3} + \frac{1}{3} + 0 = 1 \leq 1 \\
&(t=22, q=19) && \frac{1}{3} + 0 + \frac{1}{3} + \frac{1}{3} + 0 = 1 \leq 1 \\
&\vdots && \vdots
\end{aligned}$$

The verification process will always be the same, and we only assert that the remaining inequalities of the machine state are also satisfied. One can see that the machine will not be completely ramped up. However, the machine $m = 2$ starts a ramp-down in period 20 but only for one isolated period. There are 9 ramp-down periods later. It is impossible to realize an isolated ramping by a break: a ramp-down, realized by a break, always consists of 10 consecutive periods of state ramp-down. Thus, each convex combination of those solutions has at least 10 consecutive periods of state ramp-down. Thus, this solution cannot be represented in $\mathcal{P}_{LP}^{B}$.

This observation only confirms that the formulation, using breaks, is a tighter description of the set of the feasible integer solutions of Definition2.1.3, even if the energy prices are nonnegative.

Corollary 3.2.10. *The break-based formulation* (3.10b)–(3.10h) *provides a stronger description of the integer feasible solutions of Problem 2.2 than the state-based formulation* (3.1b)– (3.1l), *even if the energy prices are non-negative.*

3.2.4 Model Extensions

The break-based formulation (3.10b)–(3.10h) only includes a minimum number of necessary conditions to exclude invalid integer solutions. Additional constraints can be used to tighten its linear relaxation.

Linear Ordering Subproblem

The linear ordering problem is a subproblem of the job-shop scheduling problem with flexible energy prices and time windows.

In the linear ordering problem, we are given a digraph $D = (V, A)$. The aim is to find a total ordering of the vertices of D such that the number of backward arcs $(v, w) \in A$, which start at a node v with a higher ordering label l_v and point to a node with a lower ordering label l_w, is minimized. In our application, the vertices of the directed graph are the tasks processed by machine $m \in M$, and the computed line describes the execution order of the tasks.

It is valid to extend the problem formulation by additional variables and inequalities that are necessary to describe the integer solutions of the corresponding linear ordering problem. The linear ordering problem and some inequalities of the linear ordering problem are, among others, discussed in [GJR84, GJR85]. The linear ordering problem requires additional indicator-variables $p^{i,l}_{j,k} \in \{0,1\}$ for each pair of tasks $(j,k),(i,l) \in O^M_{|m}$, $(j,k) \neq (i,l)$, executed on the same machine $m \in M$ to describe the execution order of the tasks (j,k) and (i,l). The indicator variables' meaning is defined by

$$p^{i,l}_{j,k} = \begin{cases} 1, & \text{if } (j,k) \text{ starts before } (i,l), \\ 0, & \text{otherwise.} \end{cases}$$

The combination of all indicator variables describes the execution order of the tasks processed by machine $m \in M$. The linear ordering variables must satisfy two kinds of constraints: for each $m \in M$ and for each pair of task $(j,k),(i,l) \in O^M_{|m}$ with $(j,k) \neq (i,l)$ the constraint

$$p^{i,l}_{j,k} + p^{j,k}_{i,l} = 1 \tag{3.13}$$

forces the tasks (j,k) and (i,l) to either be executed in order $(j,k) \to (i,l)$ or $(i,l) \to (j,k)$. Both relations cannot hold simultaneously since both tasks are assigned to the same machine, and the machine must process the tasks non-preemptively. Furthermore, the constraint

$$p^{i,l}_{j,k} + p^{i_3,l_3}_{i,l} + p^{j,k}_{i_3,l_3} \leq 2 \tag{3.14}$$

must hold for each $m \in M$ and the pairwise distinct tasks $(j,k),(i,l),(i_3,l_3) \in O^M_{|m}$ must hold. The constraints (3.14) are the so-called no-cycle constraints. These constraints prevent the ordering variables from creating the following execution order of the tasks

$$(j,k) \to (i,l) \to (i_3,l_3) \to (j,k)$$

in fractional solutions. The linear ordering problem allows further valid constraints mentioned in [GJR84, GJR85]. However, the presented inequalities (3.13) and (3.14) describe all feasible integral solutions, and the ordering part of the tasks will not be a central topic in the following, and thus, the consideration of the additional constraints will not be further discussed.

The ordering variables define the execution order of the tasks. Therefore, the information of the ordering variables must be linked to their respective task variables.

The following inequalities link the ordering variables and the task variables by big-M constraints. Note that we can choose the big-M to equal 1.

$$\sum_{q=0}^{t-d^{pr}_{j,k}-d^{se}_{i,l}} x_{j,k,q} - \sum_{q=0}^{t} x_{i,l,q} \geq p^{i,l}_{j,k} - 1 \quad \forall\, t \in [T[_{\mathbb{Z}} \tag{3.15}$$

and

$$\sum_{q=0}^{t-d^{pr}_{i,l}-d^{se}_{j,k}} x_{i,l,q} - \sum_{q=0}^{t} x_{j,k,q} \geq p^{j,k}_{i,l} - 1 \quad \forall\, t \in [T[_{\mathbb{Z}} \,. \tag{3.16}$$

The constraints (3.15) and (3.16) are so-called **unfixed precedence constraints** of tasks running on the same machine.

Example 3.2.11. *Consider the constraints* (3.15) *and* (3.16). *Let* $(j,k),(i,l) \in O^M_{|m}$ *two distinct tasks and* $m \in M$. *Suppose the variable* $p^{i,l}_{j,k} = 1$ *is fixed. Then, by constraint* (3.13)*, the variable* $p^{j,k}_{i,l}$ *must be* 0. *The unfixed precedence constraints can be simplified as follows for* $t \in [T[_{\mathbb{Z}}$ *:*

$$\sum_{q=0}^{t-d^{pr}_{j,k}-d^{se}_{i,l}} x_{j,k,q} - \sum_{q=0}^{t} x_{i,l,q} \geq p^{i,l}_{j,k} - 1 = 1 - 1 = 0.$$

These constraints have the shape of the presented precedence constraints (3.10c) *and only differ within the additional consideration of the setup times. The unfixed precedence constraints for* $(i,l) \to (j,k)$ *reduce to*

$$\sum_{q=0}^{t-d^{pr}_{i,l}-d^{se}_{j,k}} x_{i,l,q} - \sum_{q=0}^{t} x_{j,k,q} \geq p^{j,k}_{i,l} - 1 = 0 - 1 = -1.$$

And thus, the inequality is redundant, since

$$\sum_{q=0}^{t-d^{pr}_{i,l}-d^{se}_{j,k}} x_{i,l,q} - \sum_{q=0}^{t} x_{j,k,q} \geq -1$$

always holds for a feasible solution $(x, z^{st}, z^{rd,ru}) \in \mathcal{P}^B$.

The tasks $(j,k),(i,l)$ are processed by the same machine. Thus, the setup durations of the later task must be considered within this precedence relation. The constraints get active if the precedence order is chosen, e.g., $p^{i,l}_{j,k}$ is fixed to either 0 or 1.

The unfixed precedence constraints add the information of the disjunctive graph to the problem formulation. Otherwise, the information of the unfixed precedence constraints often cannot be detected within the problem structure provided by the initial variables and constraints of (3.10b)–(3.10i). By integrating the unfixed precedence constraints into the initial formulation, the information is present in the formulation, and many implemented techniques can exploit the new information. However, the number of unfixed precedence constraints is $\mathcal{O}(|O|^2 \cdot T)$. Thus, their number would dominate the size of the model.

Within our implementation, we treat the unfixed precedence constraints implicitly, see Section 4.3.

Corollary 3.2.12. *Let* $m \in M$ *be one machine. The constraints* (3.15) *and* (3.16) *in combination with the inclusion of the linear ordering variables* $p^{i,l}_{j,k}$ *for the pairwise distinct pairs* $(j,k),(i,l) \in O^M_{|m}$ *are a valid model extension for* $\mathcal{P}^S$.

Corollary 3.2.13. *Let* $m \in M$ *be one machine. The constraints* (3.15) *and* (3.16) *in combination with the inclusion of the linear ordering variables* $p^{i,l}_{j,k}$ *for the pairwise distinct pairs* $(j,k),(i,l) \in O^M_{|m}$ *are a valid model extension for* $\mathcal{P}^B$.

Knapsack Constraints Limiting the Total Duration of the Breaks

0/1-Knapsack constraints are well-known constraints of the form

$$\sum_{i=1}^{n} a_i x_i \leq b$$
$$x \in \{0,1\}^n$$

with non-negative $a \in \mathbb{Z}^n$ and $b \in \mathbb{Z}$ and describe the combinatorial problem called **knapsack problem**. Combinatorial algorithms and solution strategies to solve this problem efficiently are mentioned in [KV12]. The constraints describe that only a subset $I \subset [n[_{\mathbb{Z}}$ fits into the knapsack with size b. Although we consider scheduling problems with machines instead of a knapsack, a similar structure could be detected within our problem setting: Which combination of breaks still allows the setting up and processing of all tasks on the respective single machine?

All tasks $(j,k) \in O^M_{|m}$ must be processed within the time window $[T[_{\mathbb{Z}}$ by machine $m \in M$. The setup and processing of the tasks $(j,k) \in O^M_{|m}$ requires $\sum_{(j,k)\in O^M_{|m}} (d^{se}_{j,k} + d^{pr}_{j,k})$

periods. Thus, the machine m offers only a limited number of periods for breaks and standby, which can be computed by:

$$T_B^m - \sum_{(j,k)\in O^M_{|m}} (d^{se}_{j,k} + d^{pr}_{j,k}).$$

Now, the idea is to formulate constraints limiting the length of the used breaks on a machine by knapsack constraints. Therefore, the apparent idea is to aggregate the constraints (3.10d), (3.10e), (3.10f) for a certain interval to limit the size of the breaks affected by this interval. Then, we collect the information on tasks that need to be finished in the considered interval to reduce the right-hand side of the generated knapsack constraints. Then, the resulting constraints describe a knapsack problem.

Theorem 3.2.14 (Knapsack by aggregation). *Let $m \in M$ be a machine. The aggregation of* (3.10d)*,* (3.10e)*,* (3.10f) *of machine m for all $t \in [l, r[_{\mathbb{Z}} \subseteq [T_+[_{\mathbb{Z}}$ lead to the knapsack constraint*

$$\sum_{(t_0,t_1)\in B_m} \pi^B_{(t_0,t_1)} z^{rd,ru}_{m,t_0,t_1} + \sum_{q=l}^{r} \Big(z^{st}_{m,q} + \sum_{(j,k)\in O^M_{|m}} \pi^T_{j,k,q} x_{j,k,q} \Big) \leq r - l \tag{3.17}$$

with

$$\pi^B_{t_0,t_1} = \max\Big\{0, \min\{r, t_1\} - \max\{l, t_0\}\Big\} \quad \forall\ (t_0, t_1) \in B_m$$

and

$$\pi^T_{j,k,q} = \max\Big\{0, \min\{q + d^{pr}_{j,k}, r\} - \max\{q - d^{se}_{j,k}, l\}\Big\}, \ \forall\ (j,k) \in O \text{ and } q \in [T[_{\mathbb{Z}}\ .$$

Proof. The aggregation of the constraints (3.10d), (3.10e), (3.10f) for all $t \in [l, r[_{\mathbb{Z}}$ lead to the constraint (3.17). The break $(t_0, t_1) \in B_m$ participates $\max\Big\{0, \min\{r, t_1\} - \max\{l, t_0\}\Big\}$ times within the aggregated constraints, since the intervals and the breaks are built exclusively the period r, respectively t_1. The standby variable $z^{st}_{m,t}$ only appears in constraint (3.10d) with machine index m and period t. Thus, its coefficient is always 1, if $t \in [l, r[_{\mathbb{Z}}$ holds. Similar to the coefficient of the breaks, the coefficient of a task variable can be computed. The right-hand side $r - l$ equals the number of aggregated constraints with right-hand side 1. Note that the constraint for $t = r$ is not included within the aggregation.

This validity of the constraint (3.17) follows from the validity of a conic combination of valid constraints. □

Clearly, the knapsack constraints (3.17) can also be generated by commercial solvers. To that end, the solver has to detect the substructure of the time windows of the tasks in combination with aggregation of the respective constraints. This could be ineffective, and our preparation for this step does not disturb the solution process.

The following part considers different aggregation strategies and presents the respective knapsack constraints. After adding the knapsack constraints to the break-based formulation, so-called **knapsack covers** can strengthen the formulation in a second step. Algorithms for detecting strongly violated knapsack cover constraints are introduced in [Bal75, BZ78].

We will present some knapsack constraints, describing the limited duration of breaks in specific intervals. The first knapsack constraint considers $l = -d^{rd}_m$ and $r = T + d^{ru}_m$ and thus, limits the length of breaks within the complete time window. The associated maximal knapsack constraint can be generated by lifting the other variables into the constraints, using the coefficients proposed in Theorem 3.2.14.

Theorem 3.2.15 (Knapsack constraint for the maximum duration of breaks). *Let $m \in M$ be one machine. Then, the knapsack constraint*

$$\sum_{(t_0,t_1)\in B_m} (t_1 - t_0) \cdot z^{rd,ru}_{m,t_0,t_1} + \sum_{t\in[T[_{\mathbb{Z}}} z^{st}_{m,t} \leq T^m_B - \sum_{(j,k)\in O^M_{|m}} (d^{se}_{j,k} + d^{pr}_{j,k}). \tag{3.18}$$

is valid for $\mathcal{P}^B$.

Proof. Consider the constraint (3.17) and $[l, r[_{\mathbb{Z}} = [-d^{rd}_m, T + d^{ru}_m[_{\mathbb{Z}}$. The coefficients can be computed as follows:

$$\pi^B_{t_0,t_1} = \max\Big\{0, \min\{r, t_1\} - \max\{l, t_0\}\Big\} = t_1 - t_0 \quad \forall\ (t_0, t_1) \in B_m$$

and for each $(j,k) \in O^M_{|m}$ and $q \in [a_{j,k}, f_{j,k}[_{\mathbb{Z}}$:

$$\pi^T_{j,k,q} = \max\Big\{0, \min\{q + d^{pr}_{j,k}, r\} - \max\{q - d^{se}_{j,k}, l\}\Big\} = d^{pr}_{j,k} + d^{se}_{j,k}$$

and $\pi^T_{j,k,q} = 0$ for all $q \in [T[_{\mathbb{Z}} \setminus [a_{j,k}, f_{j,k}[_{\mathbb{Z}}$.

Moreover, each task $(j,k) \in O^M_{|m}$ has to be processed within $[l, r[_{\mathbb{Z}}$. Therefore, the equation

$$\sum_{(j,k)\in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k}) x_{j,k,q} = \sum_{(j,k)\in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k})$$

holds, and thus, we obtain the constraint

$$\sum_{(t_0,t_1)\in B_m} (t_1 - t_0) \cdot z^{rd,ru}_{m,t_0,t_1} + \sum_{t\in[T[_{\mathbb{Z}}} z^{st}_{m,t} \leq T^m_B - \sum_{(j,k)\in O^M_{|m}} (d^{se}_{j,k} + d^{pr}_{j,k}).$$

□

The constraint (3.18) limits the size of initial, middle and final breaks by the same bound. To that end, we could derive knapsack constraints for the initial, middle, and final parts of the time windows in detail.

Theorem 3.2.16 (Knapsack constraint for initial breaks)**.** *Let $m \in M$ be one machine. The maximal length of an initial break belonging to m is given by*

$$\min\Big(\min\{f_{j,k} - 1 - d^{se}_{j,k} \mid (j,k) \in O^M_{|m}\}, \max\{f_{j,k} - 1 + d^{pr}_{j,k} \mid (j,k) \in O^M_{|m}\} - \sum_{(j,k)\in O^M_{|m}} (d^{se}_{j,k} + d^{pr}_{j,k})\Big) + d^{rd}_m = L + d^{rd}_m$$

and the respective knapsack constraint

$$\sum_{(t_0,t_1)\in B_m:\ t_0 = -d^{rd}_m} (t_1 - t_0) \cdot z^{rd,ru}_{m,t_0,t_1} + \sum_{t=0}^{L} z^{st}_{m,t} \leq L + d^{rd}_m. \tag{3.19}$$

is a valid constraint for $\mathcal{P}^B$.

Proof. Let $m \in M$ be one machine. The machine must be already ramped up before the first task starts its setup. The latest possible setup of all task $(j,k) \in O^M_{|m}$ can be computed as

$$\min\{f_{j,k} - 1 - d^{se}_{j,k} \mid (j,k) \in O^M_{|m}\}.$$

A longer initial break would prevent at least the first task from completing the processing before its due date.

Furthermore, the machine must be active such that all tasks $(j,k) \in O^M_{|m}$ can finish their setup and processing. The processing of all the tasks must be finished in the period

$$\max\{f_{j,k} - 1 + d^{pr}_{j,k} \mid (j,k) \in O^M_{|m}\}.$$

Thus, the latest possible start of the complete processing and setup sequence is in period

$$\max\{f_{j,k} - 1 + d^{pr}_{j,k} \mid (j,k) \in O^M_{|m}\} - \sum_{(j,k)\in O^M_{|m}} (d^{se}_{j,k} + d^{pr}_{j,k}).$$

Thus, all initial breaks cannot exceed $L + d^{rd}_m$. □

Analogously, a bound for final breaks can be derived.

Theorem 3.2.17 (Knapsack constraint for final breaks)**.** *Let $m \in M$ be one machine. The maximal length of a final break is given by*

$$R = T + d^{ru}_m - \max\Big(\max\{a_{j,k} + d^{pr}_{j,k} \mid (j,k) \in O^M_{|m}\}, \min\{a_{j,k} - d^{se}_{j,k} \mid (j,k) \in O^M_{|m}\} + \sum_{(j,k)\in O^M_{|m}} (d^{se}_{j,k} + d^{pr}_{j,k})\Big)$$

and the knapsack constraint

$$\sum_{(t_0,t_1)\in B_m:\ t_1 = T^m_B} (t_1 - t_0) \cdot z^{rd,ru}_{m,t_0,t_1} + \sum_{t=R}^{T} z^{st}_{m,t} \leq T^m_B - R. \tag{3.20}$$

is a valid constraint for $\mathcal{P}^B$.

The knapsack constraints considering middle breaks are described by the following theorem.

Theorem 3.2.18 (Knapsack constraint for inner breaks)**.** *Let $m \in M$ one machine. In addition, the earliest start of a setup and the latest finish of a processing is described by*

$$L = \min\{a_{j,k} - d^{se}_{j,k} \mid (j,k) \in O^M_{|m}\}$$
$$U = \max\{f_{j,k} - 1 + d^{pr}_{j,k} \mid (j,k) \in O^M_{|m}\}.$$

Then, the knapsack constraints

$$\sum_{(t_0,t_1)\in B_m:\ L\le t_0\le t_1\le U} (t_1 - t_0)\cdot z^{rd,ru}_{m,t_0,t_1} \le U - L - \sum_{(j,k)\in O^M_{|m}} (d^{se}_{j,k} + d^{pr}_{j,k}). \tag{3.21}$$

is a valid constraint for $\mathcal{P}^B$.

Those constraints can be strengthened by including further variables with positive coefficients. However, those constraints describe the maximum length of initial, middle, and final breaks.

Another interesting interval is provided by time windows of tasks.

Theorem 3.2.19 (Knapsack constraints for local time windows)**.** *Let $m \in M$ one machine and $(j,k) \in O^M_{|m}$ one task processed by machine m. For $l = a_{j,k} - d^{se}_{j,k}$ and $r = f_{j,k} + d^{pr}_{j,k}$, we derive the knapsack constraint*

$$\sum_{(t_0,t_1)\in B_m} \pi^B_{(t_0,t_1)} z^{rd,ru}_{m,t_0,t_1} + \sum_{q=l}^{r}\Big(z^{st}_{m,q} + \sum_{(i,l)\in O^M_{|m}\setminus\{(j,k)\}} \pi^T_{i,l,q} x_{i,l,q}\Big) \le r - l - (d^{se}_{j,k} + d^{se}_{j,k}) \tag{3.22}$$

with

$$\pi^B_{t_0,t_1} = \max\Big\{0, \min\{r,t_1\} - \max\{l,t_0\}\Big\} \quad \forall\ (t_0,t_1)\in B_m$$

and

$$\pi^T_{j,k,q} = \max\Big\{0, \min\{q+d^{pr}_{j,k}, r+d^{pr}_{j,k}-1\} - \max\{q-d^{se}_{j,k}, l-d^{se}_{j,k}\}\Big\} \quad \forall\ (j,k)\in O^M_{|m}\ q\in [T[_{\mathbb{Z}}\ .$$

For each $(j,k) \in O^M_{|m}$ the constraint (3.22) is a valid constraint of $\mathcal{P}^B$.

The validity follows by 3.2.14 and the fact that

$$\sum_{q=l}^{r} \pi^T_{j,k,q} x_{j,k,q} = \sum_{q=l}^{r} (d^{se}_{j,k} + d^{se}_{j,k})\cdot x_{j,k,q} = d^{se}_{j,k} + d^{se}_{j,k}$$

holds. Section 4.1 will reuse this information when discussing the presolving and propagation techniques.

A further class of valid constraints is the class of knapsack conditions describing the relation between a ramp-up and the ramp-down on two distinct machines.

Theorem 3.2.20. *Let $(j,k) \in O$ be one task processed by machine $m \in M$ and $(j,l) \in O$ a succeeding task processed by machine $m_2 = m_{j,l}$ with $k < l \le |O^J_{|j}| - 1$. Then, the knapsack constraint*

$$\begin{aligned}\sum_{(t_0,t_1)\in B_m: t_0=-d^{rd}_m} & (t_1 - t_0) z^{rd,ru}_{m,t_0,t_1} \\ & + \sum_{(t_0,t_1)\in B_{m_2}: t_1 = T^m_B} (t_1 - t_0) z^{rd,ru}_{m_2,t_0,t_1} \\ & \le T - (d^{se}_{j,k} + \sum_{l_3=k}^{l} d^{pr}_{j,l_3}) + d^{rd}_m + d^{ru}_{m_2}\end{aligned} \tag{3.23}$$

is a valid constraint of $\mathcal{P}^B$.

Proof. The task $(j,k) \in O^J_{|j}$ needs to be processed before task $(j,l) \in O^J_{|j}$ as $k < l$ holds. An initial break belonging to machine $m = m_{j,k}$ and a final break belonging to machine $m_2 = m_{j,l}$ need to allow the completion of the setup and processing of (j,k), respectively (i,l), in each feasible solution of $\mathcal{P}^B$. The entire sequence of tasks from task (j,k) to (j,l) needs $(d^{se}_{j,k} + \sum_{l_3=k}^{l} d^{pr}_{j,l_3})$ periods to complete setup and processing. Thus, the remaining number of periods from the $T + d^{rd}_m + d^{ru}_{m_2}$ can be used by initial breaks before (j,k) on machine m and final breaks on machine m_2 after (j,l). Thus, the constraint is a valid constraint of $\mathcal{P}^B$. □

More constraints, describing feasible combinations of breaks on different machines, are possible by even greater exploitation of the problem structure. Note that constraints like (3.23) become weaker when the time window gets larger, and the gain from introducing those constraints decreases. The presented knapsack constraints are added initially to the problem formulation, such that classical presolving techniques can use the additional information to eliminate breaks.

Chapter 4

Problem-Specific Solution Strategies

4.1 Problem Reductions

This section considers problem size reduction techniques for the proposed break-based formulation (3.10a)–(3.10h). As mentioned before, the problem (3.10a)–(3.10h) is a time-indexed formulation and suffers from its large number of variables. There are T many variables for each task. Moreover, in each feasible solution, exactly one variable per task can be non-zero. In addition, each machine has at most $\mathcal{O}(T^2)$ many break variables. Even if T many breaks are used in an optimum solution, there are $T \cdot (T-1)$ many breaks fixed to zero. Some of these breaks can be detected and eliminated initially. Moreover, the limitation of the number of breaks leads to a more precise description of the objective of the optimal primal objective value by fractional solutions. This is reasoned by the fact that fewer breaks allow fewer combinations in fractional solutions.

The initial preparation of the integer program is called presolving or preprocessing. Presolving reductions are one of the most important components in MILP solvers [GKM$^+$15]. Classical preprocessing techniques detect dominating columns, do bound tightening, use conflict analysis, and detect implied bounds. The goal of presolving is to reduce the problem size by reducing the number of variables and the number of constraints.

General purpose presolvers, however, must detect those special substructures from the MILP, which often requires computationally expensive aggregations of many constraints etc. Typically, the preprocessing needs to detect special substructures to work efficiently.

The device of problem-specific presolving techniques improves the solution algorithm since the expensive comparisons of different variables and constraints by classic problem reductions are not necessary anymore. An overview of often useful presolving techniques is provided in [ABG$^+$20]. More scheduling-related presolving rules are considered in [BJS94, BS15].

4.1.1 Presolve Reductions for ILP

Before introducing the problem-specific presolving reductions, we will briefly present some key presolving techniques implemented in many commercial solvers. We consider a general integer program with binary variables in the form:

$$\min\{c^\top x \mid Ax \leq b, x \in \{0,1\}^n\} \tag{4.1}$$

with $c \in \mathbb{Q}^n$, $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$ and $n, m \in \mathbb{N}$.

Redundant Rows

One constraint $A_{i,:}x \leq b_i$ of (4.1) is called redundant if the integer feasible region stays the same if the constraint is removed from $Ax \leq b$. The decision problem of whether a constraint is redundant is an NP-complete problem.

Example 4.1.1. *Suppose the decision problem, whether a constraint is redundant, is in the complexity class P. Then, the decision problem, whether our scheduling problem has a solution with objective val $< k$, can be solved in polynomial time since we can add the problem constraint objective $\leq k$ and check whether the objective-constraint is redundant.*

In the case of an ILP, removing redundant constraints can weaken the LP-relaxation. In the case of the presented job-shop scheduling problem, the constraints (3.18) are redundant since these constraints do not cut off any of the integral solutions of $\mathcal{P}^B$. Consequently, one is interested in detecting redundant constraints of the LP-relaxation of a MIP. Removing those constraints from the MIP $\min\{c^\top x \mid Ax \leq b, x \in \{0,1\}^n\}$ will not lead to a weaker LP-relaxation. The solution set of the LP-relaxation of the MIP is not changed, and considering fewer constraints leads to speed-up since these constraints need not be verified or considered during the solution process. The decision problem of whether a constraint $A_{i,:}x \leq b_i$ is redundant for the LP-relaxation (of a MILP) can be solved in polynomial time by solving the LP

$$b_i \geq \max\Big\{A_{i,:}x \mid A_{j,:}x \leq b_j \ \forall\, j \in \{1,\dots,m\} \setminus \{i\},\ x \in [0,1]^n\Big\}.$$

Within the review of modern presolving techniques in [ABG+20], the detection of redundant rows is mentioned to be too expensive to be helpful in practice since, for each constraint, a linear program must be solved. The authors present a more detailed descriptions of possible extensions of redundancy tests. In addition, different techniques exist to generate redundant constraints for a MIP by disaggregation of constraints [Mar01] to strengthen the LP-relaxation.

Example 4.1.2 (Aggregated and disaggregated precedence constraints). *The aggregated precedence constraint of task $(j,k),(j,k+1) \in O$ can be formulated by*

$$\sum_{t \in [T[_{\mathbb{Z}}} t \cdot (x_{j,k+1,t} - x_{j,k,t}) \geq d^{pr}_{j,k}. \tag{4.2}$$

The disaggregated form of this precedence constraint is given by

$$\sum_{q=0}^{t-d^{pr}_{j,k}} x_{j,k,q} - \sum_{q=0}^{t} x_{j,k+1,q} \geq 0, \quad t \in [T[_{\mathbb{Z}}\ .$$

Combining the precedence constraint (3.10c) *and* (3.10b) *with the integrality bounds* (3.10g) *leads to a description of the aggregated precedence constraint* (4.2).

A further way to detect redundant constraints is by comparing two rows of the linear system.

Theorem 4.1.3. *Given an ILP of the form* (4.1). *The inequality $A_{i,:}x \leq b_i$ is a redundant row of LP-relaxation of* (4.1), *if there exists a different inequality $A_{j,:}x \leq b_j$, $j \neq i$, $i,j \in [n[_{\mathbb{Z}}$, satisfying $A_{j,:} \geq A_{i,:}$ and $b_j \leq b_i$.*

Proof. The variables $x \geq 0$ are nonnegative. Since, $A_{j,:} \geq A_{i,:}$, the relation $A_{j,:}x \geq A_{i,:}x$ holds. Then,

$$A_{i,:}x \leq A_{j,:}x \leq b_j \leq b_i$$

holds. Because of $A_{j,:}x \leq b_j$, the inequality $A_{i,:}x \leq b_i$ is redundant. □

There are also valid reductions by usage of the dual problem.

Dominating Columns

The preprocessing scheme *dominated columns* exploits relations between two specific variables (columns) and is extensively presented in [GKM+15]. This presolving approach analyses the coefficients of two distinct binary variables. It can be viewed as the redundancy test of constraints of the dual system.

Definition 4.1.4 (Dominating variable). *We are given an ILP of the form* (4.1). *We say that the binary variable x_j dominates x_k, with pairwise distinct $j,k \in [n[_{\mathbb{Z}}$, if*

1. $c_j \leq c_k$
2. $A_{:,j} \leq A_{:,k}$

holds. The variable x_k is the dominated variable, while the variable x_j is the dominating variable.

Under certain conditions, one can decide that certain variables (columns) are always fixed to zero in optimal solutions. This is valid because the dominating variable is less restrictive and cheaper. Therefore, we cite the following result from [GKM+15].

Theorem 4.1.5. *We are given an ILP of the form* (4.1) *and the two distinct binary variables x_j and x_k, $j \neq k$, $j, k \in [n[_{\mathbb{Z}}$. Let $\overline{x}$ be a feasible solution of ILP* (4.1)*. Further let x_j dominating x_k. For $0 < \alpha \in \mathbb{R}$, we define x^* by*

$$x_i^* = \begin{cases} \hat{x}_i + \alpha & i = j, \\ \hat{x}_i - \alpha & i = k, \\ \hat{x}_i & else. \end{cases}$$

If $0 \leq x_i^* \leq 1$ holds for each $i \in \{1, \dots, n\}$, then the objective of x^* is not worse than the one of $\hat{x}$.

The proof of this theorem is in [GKM$^+$15] and is based on evaluating the left-hand side of each constraint $A_{r,:}x \leq b_r$. The construction leads to an (integer) feasible solution, and because of the relation of the objective coefficients, a constructed solution x^* cannot be worse than $\hat{x}$. The important result is the following one, also published in [GKM$^+$15].

Theorem 4.1.6. *We are given an ILP of the form* (4.1) *with two distinct binary variables x_j and x_k, while x_j dominates x_k. If $\overline{x}$ is an optimal solution with $x_k = 1$, then there exists an optimal solution with $x_j = 1$ and $x_k = 0$.*

Again, the proof is published in [GKM$^+$15] and is based on using Theorem 4.1.5 and choosing the correct α.

Set Dominated Columns

Dominating columns only consider the relation of the two binary variables at the same time, while the set-dominated columns approach extends this approach to the domination of one column by a set of columns.

Definition 4.1.7 (Column dominating set)**.** *We are given an ILP of the form* (4.1) *with $c \in \mathbb{Z}^n$ and $A \in Q^{m \times n}$, $n, m \in \mathbb{N}$. The column $k \in \{1, \dots, n\}$ is dominated by a subset $S \subseteq \{1, \dots, n\} \setminus \{k\}$, if*

1. $A_k = \sum_{j \in S} A_j$
2. *and* $c_k \geq \sum_{j \in S} c_j$

hold.

Now, we reproduce the results of [GKM$^+$15] by transforming the theory of dominating columns to dominating sets.

Theorem 4.1.8. *We are given an ILP of the form*

$$\min \left\{ c^\top x \mid Ax = 1, x \in \{0,1\}^n \right\} \tag{4.3}$$

with $c \in \mathbb{Z}^n$ and $A \in \{0,1\}^{m \times n}$, $n, m \in \mathbb{N}$. Let column $k \in \{1, \dots, n\}$ be set-dominated by the $S \subset \{1, \dots, n\} \setminus \{k\}$. Further, let $\alpha \in \mathbb{R}$ and $\hat{x}$ be a feasible solution for the ILP. We define x^* with

$$x_i^* = \begin{cases} \hat{x}_i + \alpha & i \in S, \\ \hat{x}_i - \alpha & i = k, \\ \hat{x}_i & else. \end{cases}$$

If $0 \leq x_i^* \leq 1$ holds for each $i \in \{1, \dots, n\}$ and x^* is feasible, then the objective of x^* is not worse than the objective of $\hat{x}$.

Proof. We are given the ILP 4.3. In addition, the column $k \in \{1, \dots, n\}$ is dominated by the set $S \subseteq \{1, \dots, n\} \setminus \{k\}$. Further, let $\alpha \in \mathbb{R}$ and $\hat{x}$ be the feasible solution of the ILP. We define x^* by

$$x_i^* = \begin{cases} \hat{x}_i + \alpha & i \in S, \\ \hat{x}_i - \alpha & i = k, \\ \hat{x}_i & \text{else.} \end{cases}$$

If $0 \leq x_i^* \leq 1$ holds, then each row $r \in \{1,\ldots,m\}$ satisfies:

$$
\begin{aligned}
A_{r,:}x^* &= \sum_{i=1}^{n} A_{r,i}x_i^* \\
&= \sum_{i\in[1,n+1[_{\mathbb{Z}}\setminus S\cup\{k\}} A_{r,i}x_i^* + \sum_{i\in S\cup\{k\}} A_{r,i}x_i^* \\
&= \sum_{i\in[1,n+1[_{\mathbb{Z}}\setminus S\cup\{k\}} A_{r,i}x_i^* + \sum_{i\in S} A_{r,i}(\hat{x}_i+\alpha) + A_{r,k}(\hat{x}_k-\alpha) \\
&= \sum_{i\in[1,n+1[_{\mathbb{Z}}\setminus S\cup\{k\}} A_{r,i}\hat{x}_i + \sum_{i\in S} A_{r,i}(\hat{x}_i+\alpha) + \Big(\sum_{i\in S} A_{r,i}\Big)(\hat{x}_k-\alpha) \\
&= \sum_{i\in[1,n+1[_{\mathbb{Z}}\setminus S\cup\{k\}} A_{r,i}\hat{x}_i + \sum_{i\in S} A_{r,i}\hat{x}_i + A_{r,k}\hat{x}_k \\
&= A_{r,:}\hat{x} = b_r.
\end{aligned}
$$

Thus, the solution x^* is also feasible. In addition, the transformation of the solution only improves the objective value since

$$
\begin{aligned}
c^\top \hat{x} &= \sum_{i\in[1,n+1[_{\mathbb{Z}}} c_i\hat{x}_i \\
&= \sum_{i\in[1,n+1[_{\mathbb{Z}}\setminus S\cup\{k\}} c_i\hat{x}_i + \sum_{i\in S} c_i\hat{x}_i + c_k\hat{x}_k \\
&\geq \sum_{i\in[1,n+1[_{\mathbb{Z}}\setminus S\cup\{k\}} c_i x_i^* + \sum_{i\in S} c_i(x_i^*+\alpha) + c_k(x_k^*-\alpha) = c^\top x^*.
\end{aligned}
$$

holds. Thus, the solution x^* is not worse than $\hat{x}$. □

The following result describes that at least one optimal solution of the optimization problem is not affected at all.

Theorem 4.1.9. *We are given an ILP of form*

$$\min\Big\{c^\top x \mid Ax = 1, x \in \{0,1\}^n\Big\}$$

with $c \in \mathbb{Z}^n$ and $A \in \mathbb{Q}^{m\times n}$, $n, m \in \mathbb{N}$. Let x_k be dominated by the set $S \subseteq \{1,\ldots,n\}\setminus\{k\}$. If there exists an optimal solution x^* with $x_k^* = 1$, then there exists also an optimal solution with $x_k = 0$ and $x_j = 1$ for each $j \in S$.

The proof is analogous to the proof of Theorem 4.1.5 while using the theorem 4.1.8. However, the computation of $S \subseteq \{1,\ldots,n\}$ is not clear. First of all, we propose to compute the set S, dominating column $k \in [n[_{\mathbb{Z}}$, by solving the integer program:

$$\min\Big\{c^\top x \mid Ax = A_k,\ x_k = 0,\ x \in \{0,1\}^n\Big\}. \tag{4.4}$$

There are three possible results of this optimization problem:

- We compute the optimal solution x^* with $c^\top x^* \leq c_k$. Then, at least one optimal solution exists to the original optimization problem with $x_k = 0$. For a minimal k-dominating variable set S with $\sum_{i\in S} A_{i,:} = A_k$, either one $x_i = 1$, with $i \in S$ or $x_k = 1$.
- There exists one optimal solution x^* with $c^\top x^* > c_k$. Then, the variable x_k cannot be fixed to zero. However, the variables x_i for $i \in S$ will not be nonnegative simultaneously.
- There exists no solution. Then, the variable x_k cannot be fixed to zero. The variable x_k also cannot be fixed to one, since (4.4) is not the original problem

$$\min\Big\{c^\top x \mid Ax = 1, x \in \{0,1\}^n\Big\}.$$

It is impossible that the optimization problem is unbounded because the variables are binary. Solving an integer linear program to decide whether one single variable can be fixed to zero is expensive. Especially if the problem is as large as the problem that one wants to solve originally.

Nevertheless, the amount of work for solving an ILP for each column, which can be as hard as solving the original problem, is too large. Therefore, we want to solve smaller and easier problems to presolve the columns.

Lemma 4.1.10. *We are given an ILP of the form* (4.3). *If column* $k \in \{1, \ldots, n\}$ *is set-dominated by* $S \subseteq \{1, \ldots, n\} \setminus \{k\}$*, then* $A_{r,i} = 0$ *holds for each column* $i \in S$ *and row* $r \in \{r \in [m+1[_{\mathbb{Z}} \mid A_{r,k} = 0\}$.

Proof. Assuming, there exists at least for one $i \in S$ with $A_{r,i} = 1$ for one $r \in \{r \in [m+1[_{\mathbb{Z}} \mid A_{r,k} = 0\}$. Then

$$\sum_{i \in S} A_{r,i} \geq 1 > 0 = A_{r,k}.$$

Thus, S is not a k-dominating set. Therefore, only columns with $A_{r,i} = 0$ holds for each $i \in S$ and $r \in \{r \in [m+1[_{\mathbb{Z}} \mid A_{r,k} = 0\}$ are necessary to build a k dominating set. □

Lemma 4.1.10 helps to identify the relevant columns for computing the k dominating set S. Thus, the column k only affects a subset $R = \{r \in [m+1[_{\mathbb{Z}} \mid A_{r,k} \neq 0\}$. Then, only the columns $I = \{i \in [n+1[_{\mathbb{Z}} \mid A_{r,i} = 0 \; \forall \; r \in [m+1[_{\mathbb{Z}} \setminus R\}$ are of interest. Therefore, one can reduce the number of columns within this presolving problem and only solve the IP

$$\min \left\{ c^\top x \mid A_{R,I} x = A_k, \; x_k = 0, \; x \in \{0,1\}^n \right\}.$$

This approach can reduce the amount of work to solve the presolving ILP. Nevertheless, one needs to solve many ILPs. To only solve the presolving problem of computing a k dominating set S with good prospects, the following rules should be followed.

- Sort the columns $i, j \in \{1, \ldots, n\}$ in descending order $c_j \geq c_i$ to sift out expensive variables in the beginning. Expensive columns $i \in \{1, \ldots, n\}$ will not participate in k dominating sets if the own objective c_i function value is already larger than c_k.
- If the number of non-zeros $\sum_{r=1}^{m} |A_{r,j}|$ of the column j is relatively small concerning the maximum number of non-zeros of a column, the optimization problem (4.4) will probably have no solution. The reason is the low number of combinations to re-generate the column by multiple columns. The corresponding presolving problem should be considered only at the end of the presolving stage or even not at all.
- If all columns nearly have the same objective value, this procedure will not be successful. The reason is that two columns of a set $S \subset \{1, \ldots, n\}$ will already exceed the objective $c_k < \sum_{i \in S} c_i \approx |S| c_k$. Then, the presolving stage could be skipped.

Remark 4.1.11. *The integrality condition* $x \in \{0,1\}$ *within the computation of*

$$\min \left\{ c^\top x \mid A_{R,I} x = A_k, \; x_k = 0, \; x \in \{0,1\}^n \right\}$$

is crucial. Consider the problem

$$\begin{array}{llllll}
x_1 + & x_2 + & & x_4 & = 1 \\
x_1 + & x_2 + & x_3 & & = 1 \\
x_1 + & & x_3 + & x_4 & = 1.
\end{array}$$

The column of x_1 *is not dominated by* $\{x_2, x_3, x_4\}$*, although*

$$0.5 \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} + 0.5 \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + 0.5 \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

holds.

Since detecting a set dominated column requires integral solutions, efficient combinatorial algorithms exploiting the combinatorial substructure must be devised to use this presolving rule efficiently, or some special substructure of the coefficient matrix needs to be detected (totally unimodular) to solve the presolving ILP efficiently. Otherwise, this presolving step would be too expensive.

Probing

Probing is a well-known presolving technique [ABG+20] and is used to detect logical implications between binary variables. One can iteratively set one binary variable x_i, $i \in \{1, \ldots, n\}$, to 0 and 1 and explore the two resulting problems. Moreover, the probing technique can also be applied again to the resulting problems. If the resulting problem for $x_i = 1$ is infeasible, the variable x_i can be fixed to 0 globally. If the resulting problem for $x_i = 0$ is infeasible, then the variable can be fixed to x_i to 1 globally. In addition, one can derive logical implications from probing iteratively on two variables x_i and x_k, $i \neq k, \; i, k \in [n[_{\mathbb{Z}} \; [m[_{\mathbb{Z}}$ as follows.

1. If $x_i = 0$ and $x_k = 0$ leads to an infeasibility, then the constraint $x_i + x_k \geq 1$ is valid.
2. If $x_i = 1$ and $x_k = 1$ leads to an infeasibility, then the constraint $x_i + x_k \leq 1$ is valid.
3. If $x_i = 1$ and $x_k = 0$ leads to an infeasibility, then the constraint $x_i \leq x_k$ is valid.
4. If $x_i = 0$ and $x_k = 1$ leads to an infeasibility, then the constraint $x_k \leq x_i$ is valid.

The order of the fixations within the probing of two variables is irrelevant, and the derived conflict and implication stay the same. However, the order in which the variables are fixed in probing is important and offers much space for improvement. Especially when the probing algorithm is aborted after a predefined number of evaluations. Thus, the probing order becomes crucial since some variables are more interesting than others.

The probing algorithm can also generate a detailed conflict graph to separate valid inequalities if the solver can compute the reason for the infeasibility. In [Sav94], one can find more information about probing possibilities within the presolving stage.

The expensive part of probing is the necessity of solving a large number of LPs. Limiting the number of iterations when resolving the probing LP is possible, but the time spent solving LP can still be large, and most implementations stop probing before this presolving consumes too much time if the success in global fixations and detected implications is too low. A state-of-the-art summary is in [ABG$^+$20].

Probing reductions often lead to additional fixations and implications. However, the entire probing process is time-consuming and combinatorial algorithms and conditions are more favorable.

4.1.2 Presolving Techniques for Task Variables

The number of task variables is always a big disadvantage when solving time-indexed models. Often, the number of task variables must be reduced to obtain models, that are solvable in acceptable time. Some very often used presolving rules are mentioned in [Bru02]. Most of these approaches are not applicable to job-shop scheduling with flexible energy prices and time windows. A more applicable propagation rule is mentioned in [BS15]. The authors eliminate variables that cannot be part of locally optimal solutions by using the reduced costs to compare the lower bound of locally feasible solutions and the incumbent solution. However, we need to exploit the problem structure to shrink the problem size initially.

The introduction of the most common presolving techniques shows us that some of the presolving rules require the solution of an LP or even a solution of an ILP. Now, we introduce combinatorial counterparts of the mentioned classical presolving rules. These rules are problem-specific presolving rules of the job-shop scheduling problem with flexible energy prices and time windows. To that end, we encode combinatorial conditions and algorithms to replace the probing and the dominating set computation.

This section is divided into two parts. In the beginning, we introduce the presolving rules affecting the task variables. The second part discusses the presolving rules affecting the break variables. Presolving rules affecting the standby variables are neglected since most obvious reductions are already done by probing on them. All mentioned presolving rules can also be applied as propagating rules within the branch-and-bound tree.

For each task $(j, k) \in O$ the time window $[a_{j,k}, f_{j,k}[_{\mathbb{Z}}$ is assumed to be the smallest interval in $[T[_{\mathbb{Z}}$ such that for each $t \in [T[_{\mathbb{Z}} \setminus [a_{j,k}, f_{j,k}[_{\mathbb{Z}}$ the corresponding task variable $x_{j,k,t}$ is fixed to zero. Note that there is the possibility that $x_{j,k,t} = 0$ is fixed for a period$t \in [a_{j,k} + 1, f_{j,k} - 1[_{\mathbb{Z}}$.

We start with trivial infeasibility conditions, which can be verified at each branch-and-bound node.

Remark 4.1.12. *Our variable reductions are also applicable as propagation rules. If variable reductions are detected in the branch-and-bound tree, then the reductions are only valid within the specific branch. Also, if there are variable reductions at the root node, then these reductions are also feasible for the specific branch, which equals the complete branch-and-bound tree.*

Infeasibility Due to Time Windows of Single Tasks

The task variables are binary variables that must satisfy the assignment constraints (3.10b) and the precedence constraints of the job sequences (3.10c). The assignment constraints (3.10b) describe that each task must be processed once. If one task cannot be processed, then the corresponding branch-and-bound node is infeasible.

Theorem 4.1.13. *Let $(j,k) \in O$ be one task. The current branch-and-bound node is infeasible if the condition*

$$f_{j,k} - a_{j,k} \leq 0 \tag{4.5}$$

is satisfied.

Proof. Consider $(j,k) \in O$, where the tasks time window satisfy the condition

$$f_{j,k} - a_{j,k} \leq 0.$$

Then, $[a_{j,k}, f_{j,k}[_{\mathbb{Z}}$ is empty and the task (j,k) cannot start processing. Thus, the assignment constraint

$$\sum_{t \in [a_{j,k}, f_{j,k}[_{\mathbb{Z}}} x_{j,k,t} = \sum_{t \in \emptyset} x_{j,k,t} = 0 \neq 1$$

is violated. Therefore, this case leads to an infeasible branch-and-bound node. □

MILP-solvers will also detect this infeasibility by solving the corresponding LP-relaxation. This combinatorial condition can be generalized to a subset of tasks processed by the same machine.

Theorem 4.1.14. *Let $S \subseteq O^M_{|m}$ be a subset of the tasks processed by machine $m \in M$. The current branch-and-bound node is infeasible if the condition*

$$\max_{(j,k) \in S} (f_{j,k} - 1 + d^{pr}_{j,k}) - \min_{(j,k) \in S} (a_{j,k} - d^{se}_{j,k}) < \sum_{(j,k) \in S} (d^{se}_{j,k} + d^{pr}_{j,k}) \tag{4.6}$$

is satisfied.

Proof. We are given the subset $S \subseteq O^M_{|m}$ of tasks processed by machine $m \in M$. The number of periods required to process and setup all tasks $(j,k) \in S$ without any idling time is

$$\sum_{(j,k) \in S} (d^{se}_{j,k} + d^{pr}_{j,k}).$$

The earliest setup of tasks $(j,k) \in S$ can start in period $\min_{(j,k) \in S}(a_{j,k} - d^{se}_{j,k})$ and the processing of all tasks $(j,k) \in S$ must be completed in period $\max_{(j,k) \in S}(f_{j,k} - 1 + d^{pr}_{j,k})$. The execution order of the tasks is not considered. Since

$$\min_{(j,k) \in S} (a_{j,k} - d^{se}_{j,k}) + \sum_{(j,k) \in S} d^{se}_{j,k} + d^{pr}_{j,k} > \max_{(j,k) \in S} (f_{j,k} - 1 + d^{pr}_{j,k})$$

holds, at least one task $(j,k) \in O^M_{|m}$ cannot complete its processing till the end of period $\max_{(j,k) \in S}(f_{j,k} - 1 + d^{pr}_{j,k})$. Thus, the current branch-and-bound node is infeasible. □

A brute force evaluation of all conditions of type (4.6) requires an exponential number of comparisons. We can reduce the number of comparisons to only $\mathcal{O}(|O^M_{|m}|^2)$ comparisons.

Theorem 4.1.15. *Let $m \in M$ be one machine. One can decide by $\mathcal{O}(|O^M_{|m}|^2)$ operations whether there exists a subset $S \subseteq O^M_{|m}$ satisfying all conditions of type* (4.6)

Proof. First, we will describe an algorithm that needs $\mathcal{O}(|O^M_{|m}|^2)$ operations to check the infeasibility conditions of the interesting subsets $S \subseteq O^M_{|m}$. In the second step, we assume that the infeasibility condition (4.6) is satisfied by one set $S \subseteq O^M_{|m}$. Then, we show that there also exists one set $S' \subseteq O^M_{|m}$, constructed by the algorithm, satisfying Condition (4.6).

Algorithm 1 describes a Greedy-like procedure to compute a subset S satisfying the infeasibility condition 4.6. The algorithm selects one task and then greedily increases the number of tasks so that the minimum starting period of the tasks does not change and the maximum processing start plus the processing duration is as small as possible.

For each task $(i,l) \in O^M_{|m}$, the algorithm creates an iteratively growing set S. Within the while loop, the set S is extended by further tasks. To extend the set S, two computations are required. The computation of (j,k) requires $\mathcal{O}(|O^M_{|m}|)$ operations. The if-clause is also verified in $\mathcal{O}(|O^M_{|m}|)$ operations. Thus, the complete algorithm runs in $\mathcal{O}(|O^M_{|m}|^2)$ operations.

Now, we prove that the algorithm works correctly. Let $S \subseteq O^M_{|m}$ satisfy the infeasibility condition (4.6). Denote

$$(i,l) = \underset{(i_3,l_3) \in S}{\text{argmin}}\ a_{i_3,l_3} - d^{se}_{i_3,l_3}$$

Algorithm 1 InfeasibilityCheck

```
procedure INFEASIBILITYCHECK
    for (i,l) ∈ O^M_{|m} do                                   ▷ (i,l) is the initial element
        S = {(i,l)}
        t_min = min{a_{j,k} − d^{se}_{j,k} | (j,k) ∈ S}
        while True do
            Q = {(i_3,l_3) ∈ O^M_{|m} \ S | a_{i_3,l_3} − d^{se}_{i_3,l_3} ≥ t_min}
            if Q ≠ ∅ then
                (i,l) = argmin{f_{i_3,l_3} + d^{pr}_{i_3,l_3} | (i_3,l_3) ∈ Q}
                S = S ∪ {(i,l)}
                if S satisfies (4.6) then
                    return infeasible
                end if
            else
                break
            end if
        end while
    end for
    return: no infeasibility detected
end procedure
```

the initial task and $S' = \{(i,l)\}$. Then, the proposed algorithm will add tasks $(j,k) \in O^M_{|m} \setminus S'$ to S' until no further task satisfying the required condition can be added. Let S'' be the first set within our algorithm containing S. The set S'' exists, since the tasks in $(j,k) \in S$ can be sorted in increasing order of $f_{j,k} + d^{pr}_{j,k}$. Then, the following equations and inequalities are valid:

$$\min_{(j,k)\in S} a_{j,k} - d^{se}_{j,k} = \min_{(j,k)\in S''} a_{j,k} - d^{se}_{j,k}, \tag{4.7}$$

$$\max_{(j,k)\in S} f_{j,k} + d^{pr}_{j,k} = \max_{(j,k)\in S''} f_{j,k} + d^{pr}_{j,k}, \tag{4.8}$$

$$\sum_{(j,k)\in S} d^{pr}_{j,k} + d^{se}_{j,k} \leq \sum_{(j,k)\in S''} d^{pr}_{j,k} + d^{se}_{j,k}. \tag{4.9}$$

Condition (4.7) holds by choice of (i,l) and the way the algorithm extends the set S'. Conditions (4.8) holds since we are considering the first iteration, where $S \subseteq S''$ and we only extend S' by the element $(j,k) \in O^M_{|m} \setminus S'$ with the smallest value $f_{i_3,l_3} + d^{pr}_{i_3,l_3}$. The condition (4.9) holds, since $S \subseteq S''$ holds. Therefore, S'' leads also to a satisfied infeasibility condition (4.6) and S'' is created by Algorithm 1. □

MILP solvers can also detect infeasibility by solving the corresponding linear program of the current branch-and-bound node. But the idea of Algorithm 1 will be reused multiple times within this thesis. We do not consider the analogous infeasibility check for job sequences. This is reasoned by the fact that we can trace back the infeasibility of a job sequence to the infeasibility condition of single tasks: after an adjustment of the processing starts of each task (j,k) and the computation of the local time window $[a_{j,k}, f_{j,k}[_{\mathbb{Z}}$ of the job sequence $j \in J$, the allowed processing starts of each task $(j,l) \in O^J_{|j}$ can be adjusted. If the complete job sequence cannot be set up and processed within a time window, then at least one task has an empty local time window. Otherwise, the adjustment of the local time windows of the job sequence is not as tight as possible. Nevertheless, the propagation of time windows by precedence constraints and detecting locally valid precedence relations are important.

Implied Precedence Constraints and Linear Ordering

Initially, the problem formulation includes the precedence constraints between consecutive tasks of the job sequences. This subsection discusses the identification of valid precedence constraints between tasks processed by the same machine to strengthen the problem formulation. In [Bru02], more approaches to detect valid implied precedence constraints are presented.
We are given two distinct tasks (j,k) and $(i,l) \in O^M_{|m}$ processed by machine $m \in M$. Each

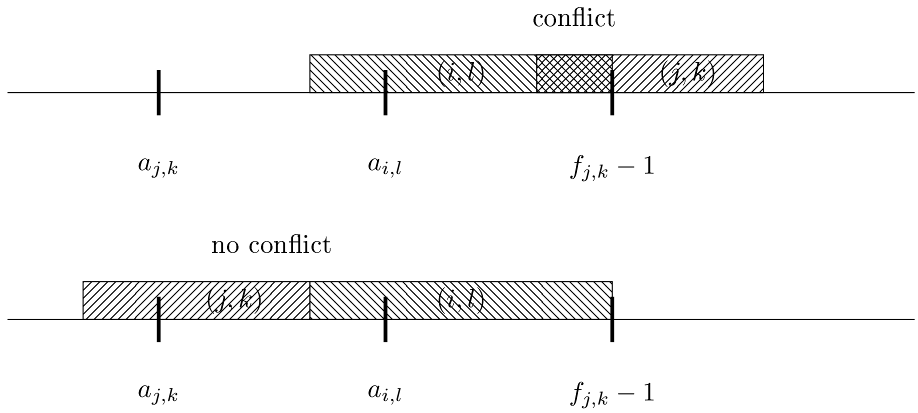


Figure 4.1: Visualization of the fixation of the execution order. The first example shows that task (i,l) will overlap with the setup of task (j,k) if the execution order $(i,l) \to (j,k)$ is fixed. The second example shows a valid fixed execution order, where the relation $(j,k) \to (i,l)$ holds, which allows both tasks to complete the processing.

task has its own time window $[a_{j,k}, f_{j,k}[_{\mathbb{Z}}$, respectively $[a_{i,l}, f_{i,l}[_{\mathbb{Z}}$. We will start with a condition for identifying valid precedence constraints. We also associate the corresponding precedence constraints with a precedence relation $(j,k) \to (i,l)$ for $(j,k),(i,l) \in O^M_{|m}$.

Definition 4.1.16. *Let $(j,k),(i,l) \in O^M_{|m}$ be two distinct tasks processed by machine $m \in M$. The precedence relation $(j,k) \to (i,l)$ are locally valid for $\mathcal{P}^B$ if the precedence relation $(i,l) \to (j,k)$ leads to an infeasibility.*

This definition is meaningful since the described property can be derived from the feasibility/ infeasibility by probing on indicator variables of the associated linear ordering problem.

Lemma 4.1.17. *Let $(j,k),(i,l) \in O^M_{|m}$ be two distinct tasks processed by machine $m \in M$. If the precedence relation $(i,l) \to (j,k)$ leads to an infeasibility, and the precedence relation $(j,k) \to (i,l)$ leads to an infeasibility of the current node, then the current node is infeasible.*

The following theorem characterizes a subset of the locally valid precedence constraints.

Theorem 4.1.18. *Let $(j,k),(i,l) \in O^M_{|m}$ be two distinct tasks processed by machine $m \in M$. The precedence relation $(j,k) \to (i,l)$ is locally valid for $\mathcal{P}^B$ if the condition*

$$a_{i,l} < f_{j,k} < a_{i,l} + d^{pr}_{i,l} + d^{se}_{j,k} \tag{4.10}$$

holds.

Proof. Let $(j,k),(i,l) \in O^M_{|m}$ be two distinct tasks processed by machine $m \in M$ such that the pair $(j,k),(i,l)$ satisfies Condition (4.10).
Suppose $\mathcal{S}^J$ is a locally feasible solution with execution order $(i,l) \to (j,k)$. The earliest start of (i,l) is $a_{i,l}$ and the latest possible start of (j,k) is $f_{j,k} - 1$. Since $f_{j,k} < a_{i,l} + d^{pr}_{i,l} + d^{se}_{j,k}$ holds, the task (j,k) cannot start processing after (i,l) and the execution order $(i,l) \to (j,k)$ is locally infeasible and there exists no feasible solution with execution order $(i,l) \to (j,k)$. □

Condition (4.10) describes that task (j,k) must start before task (i,l). This condition can be checked for each machine $m \in M$ by iterating over all distinct pairs $(j,k),(i,l) \in O^M_{|m}$ of tasks that do not have a fixed execution order.

Figure 4.1 visualizes the validity and concept of detecting implied precedence constraints. In addition to the implied precedence constraints, we can derive precedence constraints by using information from linear ordering.

Corollary 4.1.19. *Let $(j,k),(i,l),(i_3,l_3) \in O^M_{|m}$ be three pairwise distinct tasks processed by machine $m \in M$. If the tasks $(j,k),(i,l),(i_3,l_3)$ satisfy precedence relation $(j,k) \to (i,l)$ and $(i,l) \to (i_3,l_3)$, then the precedence relation $(j,k) \to (i_3,l_3)$ is locally valid for $\mathcal{P}^B$.*

Proof. The underlying linear ordering problem (4.57) and (3.14) describes all valid execution order of the pairwise distinct tasks (j,k), (i,l),$(i_3,l_3) \in O^M_{|m}$. The tasks (j,k),

(i,l), (i_3,l_3) satisfy the precedence relation $(j,k) \to (i,l)$ and $(i,l) \to (i_3,l_3)$. Within the corresponding linear ordering problem, the variables $p_{j,k}^{i,l} = 1$ and $p_{i,l}^{i_3,l_3} = 1$ are fixed.

The no-cycle inequality (3.14) fixes $p_{i_3,l_3}^{j,k} = 0$. Thus, the precedence relation $(j,k) \to (i_3,l_3)$ is locally valid for $\mathcal{P}^B$. $\square$

Also, one can use additional constraints of the linear ordering problem to derive valid fixations of precedence relations. In [GJR85], Grötschel, Jünger, and Reinelt analyzed the linear description of the integral solutions of the linear ordering problem. In addition to the no-cycle inequalities, the authors present further combinatorial constraints, which also can be used to fix the order of tasks.

However, even fixing the execution order of all tasks to the execution order of the optimal solution can still lead to fractional optimal solutions of its LP-relaxation. For more details, see Section 4.2. Thus, the theory of linear ordering, betweenness-variables and valid inequalities from disjunctive graphs are not the main focus.
Suppose a precedence relation exists between the two distinct tasks $(j,k),(i,l) \in O$ processed by machine $m \in M$. In that case, we can propagate the modifications of the time window to the preceding and succeeding tasks.

Theorem 4.1.20. *Let $(j,k),(i,l) \in O$ two distinct tasks with $(j,k) \to (i,l)$. The minimum distance between the processing start of (j,k) and the processing start of (i,l) is described by*

$$\Delta_{(j,k)}^{(i,l)} := \begin{cases} d_{j,k}^{pr}, & m_{j,k} \neq m_{i,l}, \\ d_{j,k}^{pr} + d_{j,k}^{se}, & m_{j,k} = m_{i,l}. \end{cases}$$

Then, the following precedence constraint propagation rules are valid:

$$x_{j,k,t} = 0 \quad \forall\, t \in \left\{ f_{i,l} - 1 - \Delta_{(j,k)}^{i,l)} + 1, \ldots, f_{j,k} - 1 \right\}, \tag{4.11}$$

$$x_{i,l,t} = 0 \quad \forall\, t \in \left\{ a_{i,l} \ldots, a_{j,k} + \Delta_{(j,k)}^{i,l)} - 1 \right\} \tag{4.12}$$

Proof. Let $(j,k) \in O$ and $(i,l) \in O$ two different tasks with $(j,k) \to (i,l)$. The task (i,l) is only allowed to start processing after the task (j,k) has finished its processing. If both tasks are assigned to the same machine, the task (i,l) must also complete its setup before it can start its processing. The earliest period of starting the processing of task (j,k) is period $a_{j,k}$. Thus, the earliest period of starting processing task (i,l) is $\max(a_{i,l}, a_{j,k} + \Delta_{(j,k)}^{(i,l)})$. If the processing of task (i,l) starts in or before $a_{j,k} + \Delta_{(j,k)}^{(i,l)} - 1$, then the processing of task (j,k) must start processing before its release date, which is not valid.

Analogously, the task (i,l) must start processing at least in period $f_{i,l} - 1$. The task (j,k) needs to start before (i,l). Therefore, the processing of (j,k) must be completed in period $f_{i,l} - 1 - \Delta_{(j,k)}^{(i,l)}$ to give the task (i,l) the chance to complete its processing. Any processing start of (j,k) after period $f_{i,l} - 1 - \Delta_{(j,k)}^{i,l)} + 1$ would lead to an invalid processing start of (i,l). Therefore, the processing start variable of (j,k) can be fixed to zero for $t \in \{f_{i,l} - 1 - \Delta_{(j,k)}^{(i,l)} + 1, \ldots, f_{j,k} - 1\}$. $\square$

Suppose the precedence relation $(j,k) \to (i,l)$ is valid. If the corresponding precedence constraints are integrated into the solution process, and the problem formulation, then the fixation of the task variables is redundant. The task variables cannot be used within fractional solutions. However, all other presolving rules are based on the time windows of the tasks. Thus, we must compute the local start time windows as tight as possible by doing redundant fixations.

Handling of Small Time Windows

Branchings and new (implied) precedence constraints can compress the time windows of tasks and their predecessors and successors. Then the time window $[a_{j,k}, f_{j,k}[_{\mathbb{Z}}$ is as small as possible and describes the locally valid processing starts of task $(j,k) \in O$. Note that there is the possibility that some period $t \in [a_{j,k}+1, f_{j,k}-1[_{\mathbb{Z}}$ exists where $x_{j,k,t} = 0$ is fixed.

We start with some obvious relation: if the task (j,k) is fixed to start processing in a specific period t, then the locally valid time window of task (j,k) equals $\{t\} = [a_{j,k}, f_{j,k}[_{\mathbb{Z}}$. Thus, the machine is blocked by the setup and the processing of task (j,k) within $[t - d_{j,k}^{se}, t + d_{j,k}^{pr}[_{\mathbb{Z}}$.

Theorem 4.1.21. *Let $(j,k) \in O^M_{|m}$ one task processed by machine $m \in M$. If the tasks (j,k) starts processing in period $t \in [a_{j,k}, f_{j,k}[_{\mathbb{Z}}$, then each task $(i,l) \in O^M_{|m} \setminus \{(j,k)\}$ cannot start processing in any of the respective periods $q \in [t - d^{se}_{j,k} - d^{pr}_{i,l} + 1, t + d^{pr}_{j,k} + d^{se}_{i,l}[_{\mathbb{Z}}$.*

Theorem 4.1.21 is valid since it is obviously implied by the constraints (3.10d). The small time window condition describes the cases where the task (j,k) is nearly fixed. The task can start its processing in at least two different periods. However, the valid processing starts are so close to each other that we can derive information about the machine state of the associated machine for some intermediate periods. The visualization of this presolving and propagation step is visualized in Figure 4.2.

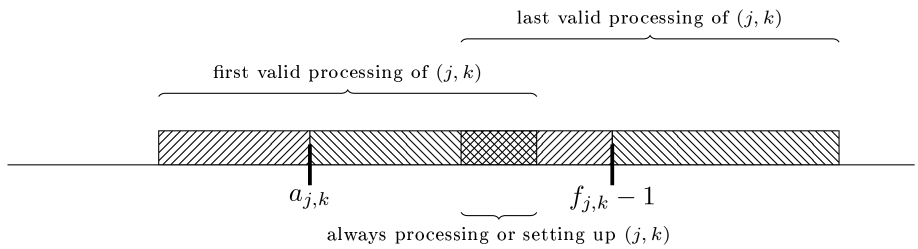


Figure 4.2: Illustration of the short time window propagation conflict period. The machine $m = m_{j,k}$ needs to process or set up the task (j,k) within the conflict period, independent of the choice of the start period of the task within the time window.

Theorem 4.1.22 (Small time windows)**.** *Let $(j,k) \in O$ be one task. If (j,k) satisfies the small time window condition*

$$a_{j,k} + d^{pr}_{j,k} > f_{j,k} - 1 - d^{se}_{j,k}, \tag{4.13}$$

then the machine $m = m_{j,k}$ needs to process or set up task (j,k) in each periods $q \in [f_{j,k} - 1 - d^{se}_{j,k}, a_{j,k} + d^{pr}_{j,k}[_{\mathbb{Z}}$.

Proof. Let $(j,k) \in O^M_{|m}$ a task processed by machine $m \in M$ that satisfies the small time window condition. If the task (j,k) starts processing in period $t \in [a_{j,k}, f_{j,k}[_{\mathbb{Z}}$, then the machine is occupied within the periods $\{t - d^{se}_{j,k}, t + d^{pr}_{j,k} - 1\}$. Then, we can consider $t = a_{j,k}$ and $t = f_{j,k} - 1$ and intersect the blocked periods on machine m. The blocked periods can be computed by

$$\{a_{j,k} - d^{se}_{j,k}, a_{j,k} + d^{pr}_{j,k} - 1\} \cap \{f_{j,k} - 1 - d^{se}_{j,k}, f_{j,k} - 1 + d^{pr}_{j,k} - 1\} = [f_{j,k} - 1 - d^{se}_{j,k}, a_{j,k} + d^{pr}_{j,k}[_{\mathbb{Z}} .$$

Thus, regardless of the choice of $t \in [a_{j,k}, f_{j,k}[_{\mathbb{Z}}$, the machine m must handle the task (j,k) while running in state *setup* or in state *processing* within the periods $q \in [f_{j,k} - 1 - d^{se}_{j,k}, a_{j,k} + d^{pr}_{j,k}[_{\mathbb{Z}}$. □

Tasks $(j,k) \in O$ satisfying the small time window condition are nearly fixed. This information can still lead to reductions. Thus, we extend the presolving schemes to nearly fixed tasks.

Theorem 4.1.23. *Let $(j,k) \in O$ one task which satisfies Condition* (4.13)*. Then, the task $(i,l) \in O^M_{|m} \setminus \{(j,k)\}$ cannot start processing in the respective periods*

$$q \in [f_{j,k} - 1 - d^{se}_{j,k} - d^{pr}_{i,l}, a_{j,k} + d^{pr}_{j,k} + d^{se}_{i,l}[_{\mathbb{Z}}$$

and the propagation scheme

$$x_{i,l,q} = 0 \quad \forall\, q \in [f_{j,k} - 1 - d^{se}_{j,k} - d^{pr}_{i,l}, a_{j,k} + d^{pr}_{j,k} + d^{se}_{i,l}[_{\mathbb{Z}} \tag{4.14}$$

is locally valid for $\mathcal{P}^B$.

Proof. Let $(j,k) \in O$ be one task satisfying the small time window condition (4.13). We consider the period $t \in [f_{j,k} - 1 - d^{se}_{j,k}, a_{j,k} + d^{pr}_{j,k}[_{\mathbb{Z}}$. Each feasible solution has to satisfy the machine state constraint

$$\sum_{(i,l) \in O^M_{|m}} \sum_{q = t - d^{pr}_{i,l} + 1}^{t + d^{se}_{i,l}} x_{i,l,q} + z^{st}_{m,t} + \sum_{\substack{(t_0,t_1) \in B_m: \\ t \in \{t_0,\dots,t_1\}}} z^{rd,ru}_{m,t_0,t_1} = 1$$

in period t. The period t satisfies $f_{j,k} - 1 - d^{se}_{j,k} \leq t \leq a_{j,k} + d^{pr}_{j,k} - 1$. Thus, the following inequalities hold:

$$t - d^{pr}_{j,k} \leq a_{j,k} - d^{se}_{j,k} \leq t \leq f_{j,k} - 1 \leq t + d^{se}_{j,k},$$

and the equation

$$\sum_{q=t-d^{pr}_{j,k}+1}^{t+d^{se}_{j,k}} x_{j,k,q} = \sum_{q=a_{j,k}}^{f_{j,k}-1} x_{j,k,q} = 1$$

holds. Therefore, for each $t \in [a_{j,k} + d^{pr}_{j,k}, f_{j,k} - d^{se}_{j,k}[_{\mathbb{Z}}$, we can fix

$$z^{st}_{m,t} + \sum_{\substack{(t_0,t_1)\in B_m:\\ t\in\{t_0,\ldots,t_1\}}} z^{rd,ru}_{m,t_0,t_1} = 0$$

$$\sum_{q=t-d^{pr}_{i,l}+1}^{t+d^{se}_{i,l}} x_{i,l,q} = 0 \quad \forall (i,l) \in O^M_{|m} \setminus \{(j,k)\}.$$

Since the variables are nonnegative, the constraints lead to a fixation of the tasks. This reduction is allowed for each $t \in [f_{j,k} - 1 - d^{se}_{j,k}, a_{j,k} + d^{pr}_{j,k}[_{\mathbb{Z}}$. Thus, the task variables for tasks $(i,l) \in O^M_{|m} \setminus \{(j,k)\}$ cannot start processing within the periods $t \in [f_{j,k} - 1 - d^{se}_{j,k} - d^{pr}_{i,l}, a_{j,k} + d^{pr}_{j,k} + d^{se}_{i,l}[_{\mathbb{Z}}$ and thus the corresponding variables can be fixed to zero. $\square$

Lemma 4.1.24. *Let $(j,k) \in O$ satisfy the small time window condition* (4.13)*. Then, the standby-variables $z^{st}_{m,t}$ for $t \in [f_{j,k} - d^{se}_{j,k}, a_{j,k} + d^{pr}_{j,k}[_{\mathbb{Z}}$ can be fixed to zero.*

Remark 4.1.25. *Suppose the task $(j,k) \in O$ starts processing in period $t \in [T[_{\mathbb{Z}}$. Then, the local time window $[a_{j,k}, f_{j,k}[_{\mathbb{Z}}$ results in $\{t_1, t_1+1\}$. Moreover, the propagation scheme by Condition 4.13 is also applicable with the highest impact.*

4.1.3 Reductions of the Break Variables

This section considers the reduction of the number of break variables. The number of breaks used in the problem formulation can be estimated by $n_M \cdot T_B^2$. Integer feasible solutions will only use a small part of all break variables to describe near-optimal solutions. One can imagine that at most T many breaks could be used by machine $m \in M$. Thus, there are $T \cdot (T-1)$ many breaks not used in one specific integral solution, and most variables are unnecessary for describing the optimal integer feasible solution. Furthermore, the large number of breaks leads to multiple (near) optimal solutions, and, generally, the machine state assignment is not unique. Therefore, the following part will discuss the reduction of the number of breaks and, hopefully, the number of near-optimal solutions.

Limiting the Length of a Break

The first presolving rule aims on computing bounds to the length of breaks and deletes the breaks exceeding those boundaries.

Within a feasible solution of the job-shop scheduling problem with flexible energy prices, each task $(j,k) \in O$ must be processed once within the time window $[T[_{\mathbb{Z}}$. Also, the machines must be ramped up before the earliest assigned task starts its setup and must be ramped down after the last task has finished its processing. We divide the breaks into four classes to obtain the strongest possible bound.

Definition 4.1.26. *Let $(t_0,t_1) \in B_m$ be one break belonging to machine $m \in M$.*

- *If $t_0 = -d^{rd}_m$ holds, then the break is called an **initial break**.*
- *If $t_0 > -d^{rd}_m$ and $t_1 < T + d^{ru}_m$ hold, then the break is called a **middle break**.*
- *If $t_0 \geq \min_{(j,k)\in O^M_{|m}} (a_{j,k} + d^{pr}_{j,k})$ and $t_1 \leq \max{(j,k) \in O^M_{|m}} (f_{j,k} - d^{se}_{j,k})$ hold, then the middle break is called an **inner break**.*
- *If $t_1 = T + d^{ru}_m$ holds, then the break is called a **final break**.*

The distinction between inner and middle breaks is necessary since an inner break shrinks the time window of a task, while a middle break need not conflict with some tasks. Each break $(t_0,t_1) \in B_m$ can be either an initial, a final or a middle break. An inner break can be combined with a task processing before or after the break. The middle break also allows the processing and setup of a task on one side of the break only. The other side of the middle break is outside of each task's time window. Nevertheless, there is a global bound for the length of each class, which was already discussed in the knapsack constraint (3.18).

Theorem 4.1.27 (Maximal length of a break)**.** *Let $(t_0, t_1) \in B_m$ one break belonging to machine $m \in M$. The break (t_0, t_1) can be eliminated if the length of the break exceeds the maximal length condition*

$$t_1 - t_0 \leq T - \sum_{(j,k) \in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k}). \tag{4.15}$$

Proof. Let $m \in M$ be one machine. Within the (expanded) time window $[-d^{rd}_m, T + d^{ru}_m[_{\mathbb{Z}}$, each task $(j,k) \in O^M_{|m}$ needs to be set up and processed once. A feasible solution to the job-shop scheduling problem with flexible energy prices needs at least two different breaks per machine: one initial break $(t_0, t_1) \in B_m$ and one final break $(t_2, t_3) \in B_m$ to cover period $t = 0$ and period $t = T$ with breaks. Thus, we derive the bound

$$t_1 - t_0 + t_3 - t_2 \leq T + d^{rd}_m + d^{ru}_m - \sum_{(j,k) \in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k}).$$

Regardless of whether (t_0, t_1) or (t_2, t_3) are final or initial breaks, the minimum length of (t_2, t_3) is $d^{rd}_m + d^{ru}_m$. Replacing the break with its bound led to

$$t_1 - t_0 \leq T - \sum_{(j,k) \in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k}).$$

Of course, this bound is also valid for middle breaks, since then, we additionally need to consider three breaks per machine and two breaks of length $d^{rd}_m + d^{ru}_m$. The resulting inequality is

$$t_5 - t_4 \leq T - \Big(\sum_{(j,k) \in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k}) \Big) - (d^{rd}_m + d^{ru}_m).$$

for middle breaks $(t_4, t_5) \in B_m$. □

This presolving and propagation scheme is based on presolving for knapsack constraints (3.18). However, we can initially compute the maximum length of the breaks and only generate the useful breaks. This presolving scheme is not useful to be reused in propagation since the length of processing and setup, as well as the time window, is constant.

Lemma 4.1.28. *Let (t_0, t_1) be one break belonging to machine $m \in M$. The break (t_0, t_1) can be eliminated if the break satisfies the condition*

$$t_1 - t_0 > T - \sum_{(j,k) \in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k}). \tag{4.16}$$

Remark 4.1.29. *The presolving scheme* (4.15) *is weak for machine $m \in M$, if*

$$\eta = \frac{\sum_{(j,k) \in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k})}{T}$$

becomes small. The enlargement of the time window decreases the value η. Then, the number of periods required by processing and setting up all tasks is comparatively small with respect to the time window. Thus, many breaks $(t_0, t_1) \in B_m$ satisfy the condition (4.15) *and thus, many breaks will not be eliminated.*

Due to the existence of release and due dates, which are not considered in (4.15), the bound on the length of initial and final breaks can be further improved.

To that end, we consider each machine independently and derive bounds by single-machine scheduling with release and due dates. Therefore, we already presented a condition to verify whether scheduling a subset of tasks is still possible within a fixed time window. Condition 4.6 detects the infeasibility of a subset of tasks that cannot complete its processing and setup within the provided time window. Now, it is to reverse the idea, and we limit the length of initial and final breaks by computing the required number of periods for setting up and processing the tasks while we maintain one side of the time window constant. To be more specific, we compute the maximum length of final and initial breaks by reducing the time windows either on the left or on the right side of the time window such that Condition 4.6 would detect an infeasibility.

Theorem 4.1.30. *Let $(t_0, t_1) \in B_m$ be one break belonging to machine $m \in M$. The break (t_0, t_1) can be eliminated if the break satisfies the condition*

$$t_0 < \max_{S \in \mathcal{P}(O^M_{|m}),\ S \neq \emptyset} \Big(\min_{(j,k) \in S} (a_{j,k} - d^{se}_{j,k}) + \sum_{(j,k) \in S} (d^{se}_{j,k} + d^{pr}_{j,k}) \Big). \tag{4.17}$$

Proof. Let $S \in \mathcal{P}(O^M_{|m})$ be one arbitrary and nonempty subset of tasks. The earliest start of a setup of one of the tasks $(j,k) \in S$ is in period $\min_{(j,k)\in S}(a_{j,k} - d^{se}_{j,k})$. Because of the release dates, no task $(j,k) \in S$ can start the setup earlier. Let $(t_0, t_1) \in B_m$ be a final break. Each feasible integer solution of $\mathcal{P}^B$ using a break (t_0, t_1) must complete the processing and set up all tasks before period t_0. The final break (t_0, t_1) starts too early if the set S satisfies the modified infeasibility condition 4.6:

$$\min\big(t_0, \max_{(j,k)\in S}(f_{j,k} - 1 + d^{pr}_{j,k})\big) - \min_{(j,k)\in S}(a_{j,k} - d^{se}_{j,k}) < \sum_{(j,k)\in S} d^{se}_{j,k} + d^{pr}_{j,k}.$$

If $t_0 \geq \max_{(j,k)\in S}(f_{j,k} - 1 + d^{pr}_{j,k})$ holds, then the infeasibility check for break (t_0, t_1) is not of interest since the maximum time window of the tasks is not changed. Therefore, we consider the case $t_0 < \max_{(j,k)\in S}(f_{j,k} - 1 + d^{pr}_{j,k})$. Then, a final break can only start in period t_0 if t_0 satisfies:

$$t_0 - \min_{(j,k)\in S}(a_{j,k} - d^{se}_{j,k}) \geq \sum_{(j,k)\in S} d^{se}_{j,k} + d^{pr}_{j,k}$$

for each $S \subseteq O^M_{|m}$. Otherwise, the condition (4.6) is not satisfied, and the current problem or branch-and-bound node is infeasible. Since the condition holds for each S, the condition must hold for the S with

$$t_0 \geq \max_{S\in\mathcal{P}(O^M_{|m})} \Big(\min_{(j,k)\in S}(a_{j,k} - d^{se}_{j,k}) + \sum_{(j,k)\in S} (d^{se}_{j,k} + d^{pr}_{j,k}) \Big).$$

□

The computation of (4.17) requires evaluating an exponential number of expressions. Therefore, we introduce for each $(j,k) \in O$ the set

$$F(j,k) := \{(i,l) \in O^M_{|m_{j,k}} \mid a_{j,k} - d^{se}_{j,k} \leq a_{i,l} - d^{se}_{i,l}\}. \tag{4.18}$$

Theorem 4.1.31. *Let $(t_0, t_1) \in B_m$ be a final break belonging to machine $m \in M$. The break (t_0, t_1) can be eliminated if the break satisfies the condition*

$$t_0 < \max_{(j,k)\in O^M_{|m}} \min_{(i,l)\in F(j,k)} (a_{i,l} - d^{se}_{i,l}) + \sum_{(i,l)\in F(j,k)} (d^{se}_{i,l} + d^{pr}_{i,l}). \tag{4.19}$$

Proof. This bound is valid since the following bound holds:

$$\begin{aligned} t_0 &\geq \max_{S\in\mathcal{P}(O^M_{|m})} \min_{(i,l)\in S} (a_{i,l} - d^{se}_{i,l}) + \sum_{(i,l)\in S} (d^{se}_{i,l} + d^{pr}_{i,l}) \\ &\geq \max_{(j,k)\in O^M_{|m}} \min_{(i,l)\in F(j,k)} (a_{i,l} - d^{se}_{i,l}) + \sum_{(i,l)\in F(j,k)} (d^{se}_{i,l} + d^{pr}_{i,l}). \end{aligned}$$

The bound is still valid since we only replace all possible subsets $S \in \mathcal{P}(O^M_{|m})$ by a subset of subsets $\{F(j,k) \mid (j,k) \in O^M_{|m}\}$. Therefore, the bound can only be weaker because we do not consider all possible subsets $S \in \mathcal{P}(O^M_{|m})$. □

This bound is computable with less effort. Moreover, we can prove that no information is lost by using (4.19) instead of (4.17).

Theorem 4.1.32. *If the break satisfies Condition* (4.17)*, then the break also satisfies Condition* (4.19).

Proof. Denote

$$S^* = \arg \max_{S\in\mathcal{P}(O^M_{|m})} \min_{(i,l)\in S} (a_{i,l} - d^{se}_{i,l}) + \sum_{(i,l)\in S} (d^{se}_{i,l} + d^{pr}_{i,l})$$

one subset of tasks defining the lower bound to the first start of a final break. Then, there exists one task $(i_3, l_3) \in S^*$ with

$$(i_3, l_3) = \arg \min_{(i,l)\in S^*} (a_{i,l} - d^{se}_{i,l}).$$

By usage of task (i_3, l_3), the set $F(i_3, l_3)$ leads to $S^* \subseteq F(i_3, l_3)$ and

$$\sum_{(i,l)\in S^*} (d^{se}_{i,l} + d^{pr}_{i,l}) \leq \sum_{(i,l)\in F(i_3,l_3)} (d^{se}_{i,l} + d^{pr}_{i,l}).$$

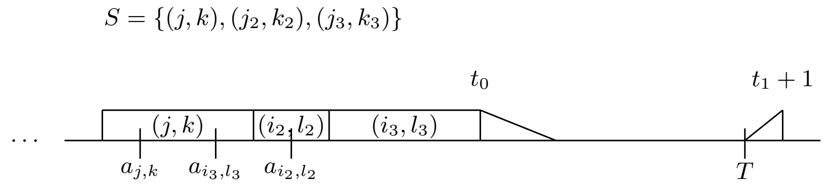


Figure 4.3: This figure shows that using sub-schedules can give better results than the pure consideration of scheduling all tasks captured in the earliest release date. By the existence of only one further task, whose release date has disappeared on the left edge of the figure, the calculated bound on the length of a final break would become too weak.

The set S^* and the set $F(i_3,l_3)$ have the same earliest start of a setup. Moreover, the length of processing and setup in $F(i_3,l_3)$ can only be larger than in S^*. Thus, we have

$$\max_{S \in \mathcal{P}(O^M_{|m})} \min_{(i,l)\in S} \Big((a_{i,l} - d^{se}_{i,l}) + \sum_{(i,l)\in S} (d^{se}_{i,l} + d^{pr}_{i,l}) \Big) \leq$$
$$\max_{(i,l)\in F(j,k)} \min_{(i,l)\in F(i_3,l_3)} \Big((a_{i,l} - d^{se}_{i,l}) + \sum_{(i,l)\in F(i_3,l_3)} (d^{se}_{i,l} + d^{pr}_{i,l}) \Big).$$

The equality of both bounds follows by $F(i_3,l_3) \in \mathcal{P}(O^M_{|m})$. □

Figure 4.3 visualizes Condition 4.19. Furthermore, a similar approach leads to upper bounds for finishing the initial ramp-up.

Theorem 4.1.33. *Let $m \in M$ be one machine. Denote*

$$D(j,k) := \{(i,l) \in O^M_{|m} \mid f_{j,k} - 1 + d^{pr}_{j,k} \geq f_{i,l} - 1 + d^{pr}_{i,l}\}$$

the set of tasks $(i,l) \in O^M_{|m}$, which are allowed to start processing later than (j,k). Then the machine needs to be ramped up at the latest in period t_1, bounded by

$$t_1 \leq \min_{(j,k)\in O^M_{|m}} \max_{(i,l)\in D(j,k)} (f_{i,l} - 1 + d^{pr}_{i,l}) - \sum_{(i,l)\in D(j,k)} (d^{se}_{i,l} + d^{pr}_{i,l}). \tag{4.20}$$

The validity of Theorem 4.1.33 can be proven similarly to Theorem 4.1.31. We presented bounds to the maximum length of final and initial breaks. However, there is also an approach to constrain the length of inner breaks.

Theorem 4.1.34. *The length of an inner break $(t_0,t_1) \in B_m$ on machine $m \in M$ is bounded by*

$$t_1 - t_0 \leq \max_{(i,l)\in O^M_{|m}} (f_{i,l} - 1 + d^{pr}_{i,l}) - \min_{(i,l)\in O^M_{|m}} (a_{i,l} - d^{se}_{i,l}) - \sum_{(i,l)\in O^M_{|m}} (d^{se}_{i,l} + d^{pr}_{i,l}). \tag{4.21}$$

Proof. Let $(t_0,t_1) \in B_m$ be an inner break. The inner break (t_0,t_1) satisfies

$$\min_{(i,l)\in O^M_{|m}} (a_{i,l} - d^{se}_{i,l}) < t_0 \text{ and } t_1 < \max_{(i,l)\in O^M_{|m}} (f_{i,l} - 1 + d^{pr}_{i,l}).$$

In each feasible integer solution of $\mathcal{P}^B$, Condition 4.6 is not satisfied by the tasks processed by machine m. Since the break (t_0,t_1) requires some space, as equal as the tasks $(j,k) \in O^M_{|m}$. Therefore, the break (t_0,t_1) cannot be used in a feasible solution if the tasks in combination with the inner break (t_0,t_1) satisfy 4.6:

$$\max_{(i,l)\in O^M_{|m}} (f_{i,l} - 1 + d^{pr}_{i,l}) - \min_{(i,l)\in O^M_{|m}} (a_{i,l} - d^{se}_{i,l}) < \sum_{(i,l)\in O^M_{|m}} (d^{se}_{i,l} + d^{pr}_{i,l}) + (t_1 - t_0).$$

Thus, we cannot detect a direct infeasibility if

$$t_1 - t_0 \leq \max_{(i,l)\in O^M_{|m}} (f_{i,l} - 1 + d^{pr}_{i,l}) - \min_{(i,l)\in O^M_{|m}} (a_{i,l} - d^{se}_{i,l}) - \sum_{(i,l)\in O^M_{|m}} (d^{se}_{i,l} + d^{pr}_{i,l})$$

holds. □

As before, the bound can be strengthened by considering also subsets $S \subseteq O^M_{|m}$.

Theorem 4.1.35. *The break* $(t_0, t_1) \in B_m$ *can be eliminated, if there exists a subset* $S \subseteq O^M_{|m}$ *satisfying*

$$t_0 > \min_{(j,k)\in S}(a_{j,k} - d^{se}_{j,k}) \text{ and } t_1 < \max_{(j,k)\in S}(f_{j,k} + d^{pr}_{j,k})$$

and

$$t_1 - t_0 > \max_{(i,l)\in S}(f_{i,l} + d^{pr}_{i,l}) - \min_{(i,l)\in S}(a_{i,l} - d^{se}_{i,l}) - \sum_{(i,l)\in S}(d^{se}_{i,l} + d^{pr}_{i,l}).$$

The validity of this bound follows directly from Theorem 4.1.34 and the fact that the assignment constraints (3.10b) can be discarded for $(j, k) \in O^M_{|m} \setminus S$. The problem results in a relaxation of the complete problem, and the fixation by Theorem 4.1.34 is applicable.

The set $S \subseteq O^M_{|m}$ can be computed by Algorithm 2.

Algorithm 2 Fixation Check For Inner Breaks

```
procedure FixationCheckForInnerBreak
    for (i,l) ∈ O^M_{|m} do                          ▷ (i,l) is the initial element
        S = {(i,l)}
        while True do
            t_min = min_{(j,k)∈S} a_{j,k} − d^{se}_{j,k}
            (j,k) = argmin_{(i3,l3)∈O^M_{|m}\S: a_{i3,l3} − d^{se}_{i3,l3} ≥ t_min} f_{i3,l3} + d^{pr}_{i3,l3}
            if argmin exists then
                if t1 − t0 > max_{(i,l)∈S}(f_{i,l} + d^{pr}_{i,l}) − min_{(i,l)∈S}(a_{i,l} − d^{se}_{i,l}) − Σ_{(i,l)∈S}(d^{se}_{i,l} + d^{pr}_{i,l}) then
                    fix break to zero
                    break
                end if
            else
                break
            end if
        end while
    end for
    return: no fixation detected
end procedure
```

Theorem 4.1.36. *Algorithm 2 works correctly and requires* $\mathcal{O}(|O^M_{|m}|^2)$ *operations.*

Note that the middle breaks, which cannot be classified as inner breaks, are not mentioned except by rule 4.15. However, propagation rule 4.1.35 can be extended to also be valid for middle breaks.

Theorem 4.1.37. *The middle break* $(t_0, t_1) \in B_m$ *belonging to machine* $m \in M$ *can be eliminated if the number of common periods of the break and the time window of the tasks, denoted by*

$$L = \min(\max_{(i,l)\in O^M_{|m}}(f_{i,l} - 1 + d^{pr}_{i,l}), t_1) - \max(\min_{(i,l)\in O^M_{|m}}(a_{i,l} - d^{se}_{i,l}), t_0),$$

satisfies

$$L > \max_{(i,l)\in O^M_{|m}}(f_{i,l} - 1 + d^{pr}_{i,l}) - \min_{(i,l)\in O^M_{|m}}(a_{i,l} - d^{se}_{i,l}) - \sum_{(i,l)\in O^M_{|m}}(d^{se}_{i,l} + d^{pr}_{i,l}). \tag{4.22}$$

Proof. Let $(t_0, t_1) \in B_m$ be one middle break satisfying (4.22). The break (t_0, t_1) and the processing interval $[\min_{(i,l)\in O^M_{|m}}(a_{i,l} - d^{se}_{i,l}), \max_{(i,l)\in O^M_{|m}}(f_{i,l} - 1 + d^{pr}_{i,l})[_{\mathbb{Z}}$ have L periods in common. If (t_0, t_1) is also an inner break, then $L = t_1 - t_0$ holds. Otherwise, there exists an inner break (q_0, q_1) with

$$q_0 = \max\big(\min_{(i,l)\in O^M_{|m}}(a_{i,l} - d^{se}_{i,l}), t_0\big)$$

$$q_1 = \min\big(\max_{(i,l)\in O^M_{|m}}(f_{i,l} - 1 + d^{pr}_{i,l}), t_1\big)$$

with $q_1 - q_0 = L$. Thus, (q_0, q_1) is an invalid inner break. The inner break is satisfying 4.1.34 and would be fixed to zero. Since $[q_0, q_1[_{\mathbb{Z}} \subseteq [t_0, t_1[_{\mathbb{Z}}$ holds, the subset of tasks can still not be processed by the machine if the break (t_0, t_1) is used. Thus, (t_0, t_1) can be fixed to zero. $\square$

However, most middle breaks that are not inner breaks could be classified as forbidden or irrelevant.

Irrelevant and forbidden Breaks

This part will discuss the detection and elimination of irrelevant breaks. An irrelevant break $(t_0, t_1) \in B_m$ describes a sequence of ramping-down, offline periods and ramping-up that will not be used in optimal solutions. Therefore, we need to define a meaningful integer feasible solution.

Definition 4.1.38 (Meaningful integer feasible solution)**.** *We call the feasible integer solution* $(\mathcal{S}^J, \mathcal{S}^M) \leftrightarrow (x, z^{st}, z^{rd,ru}) \in \mathcal{P} \cap \mathbb{Z}^{|O| \times n_M \cdot T \times \sum_{m \in M} |B_m|}$ *a* ***meaningful integer feasible solution****, if the machine state assignment of* $\mathcal{S}^M$ *is optimal for fixed* $\mathcal{S}^J$*. Otherwise, the solution is called a* ***non-meaningful integer feasible solution****.*

A meaningful integer feasible solution $(\mathcal{S}^J, \mathcal{S}^M)$ is characterized by the property of optimal machine state assignment for a fixed schedule S^J.

Figure 4.4 shows a partial example of a non-meaningful integer feasible solution. The energy prices are assumed to be nonnegative and constant. The energy demand of the machine within the machine states is also assumed to be positive. The first approach is to

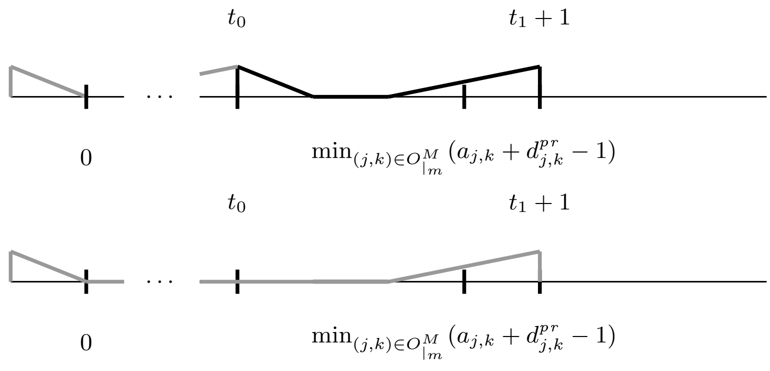


Figure 4.4: This figure shows two possible solutions for breaks at the beginning of the time window. While the first subfigure uses two breaks to cover the periods $-d^{rd}_m$ to t_1, the second solution only uses one break. In the case of only positive energy prices, the second solution, which avoids the additional ramping, will lead to a better objective value. If the energy prices are also negative, the extra ramping can be used to buy and use energy for a negative consumption price, and the first solution with two breaks can be the better one.

characterize forbidden breaks in case of nonnegative energy prices P_t, $t \in [T[_{\mathbb{Z}}$.

Theorem 4.1.39 (Forbidden breaks)**.** *Let* $(t_0, t_1) \in B_m$ *be one break belonging to machine* $m \in M$*. If the break* (t_0, t_1) *satisfies the condition*

$$-d^{rd}_m < t_0 < d^{ru}_m \text{ or } T - d^{rd}_m < t_1 < T + d^{rd}_m,$$

then the break (t_0, t_1) *can be eliminated.*

A forbidden break prevents the machine from using an initial or a final break. Since the constraints (3.10e) and (3.10f) enforce the usage of a final and an initial break, the forbidden break cannot be used. Forbidden breaks can be detected in presolving. Their number is not influenced by branching or propagation until breaks are fixed to one. Further, we can identify breaks that cannot appear in optimal solutions. Those breaks should not be generated initially.

Theorem 4.1.40 (Irrelevant breaks for nonnegative energy prices)**.** *Let* $(t_0, t_1) \in B_m$ *be one break belonging to machine* $m \in M$*.*

1. *If the break satisfy*
$$d_m^{ru} \leq t_0 \leq \min_{(j,k)\in O^M_{|m}} (a_{j,k} - d^{se}_{j,k}) \tag{4.23}$$
and the energy prices additionally satisfy $P_t > 0$ for all $t \in [t_1[_{\mathbb{Z}}$, then the break can be eliminated.
2. *If the break satisfies*
$$\max_{(j,k)\in O^M_{|m}} (f_{j,k} - 1 + d^{pr}_{j,k}) \leq t_1 < T - d^{rd}_m, \tag{4.24}$$
and the energy prices additionally satisfy $P_t > 0$ for all $t \in [T[_{\mathbb{Z}} \setminus [t_0[_{\mathbb{Z}}$, then the break can be eliminated.

Proof. Without loss of generality, we assume that the initial phase only consists of two breaks. Further breaks and standby phases can also be considered in an iterative procedure.

Let $(x, z^{st}, z^{rd,ru}) \in \mathcal{P}^B$ an integer feasible solution and $(t_2, t_3) \in B_m$ be the first break belonging to machine $m \in M$ satisfying $d_m^{ru} \leq t_2 \leq \min_{(j,k)\in O^M_{|m}} (a_{j,k} - d^{se}_{j,k})$ and $z^{rd,ru}_{m,t_2,t_3} > 0$.

Since the break (t_2, t_3) is not an initial break, there exists an initial break $(t_0, t_1) \in B_m$, with $z^{rd,ru}_{m,t_0,t_1} > 0$, and some standby periods completing the initial phase before break (t_2, t_3). Then, the following inequalities hold:

$$\begin{aligned}
&\hat{d}_{m,t_0,t_1} + \sum_{t=t_1}^{t_2-1} P_t \cdot D^{st}_m + \hat{d}_{m,t_2,t_3} = \\
&\sum_{q=t_3-d^{ru}_m}^{t_3-1} P_t D^{ru}_m + \sum_{q=t_3}^{t_0-1} P_t \cdot D^{st}_m + \sum_{q=t_0}^{t_0+d^{rd}_m-1} P_t D^{rd}_m + \sum_{q=t_1-d^{ru}_m}^{t_1} P_t D^{ru}_m \\
&\geq \sum_{q=t_1-d^{ru}_m}^{t_1} P_t D^{ru}_m \\
&= \hat{d}_{m,t_2,t_1}.
\end{aligned}$$

Thus, the solution $(x, z^{st}, z^{rd,ru})$ can be improved by replacing the sequence (t_0, t_1), standby from t_1 to $t_2 - 1$ and (t_2, t_3) by the break (t_0, t_3), when the energy prices are nonnegative for $t \in [t_3[_{\mathbb{Z}}$. Thus, an optimal solution always exists that does not use break (t_2, t_3). Thus, there exists an integral feasible solution not using break (t_2, t_3), and we can fix $z^{rd,ru}_{m,t_2,t_3} = 0$. $\square$

This presolving rule eliminates breaks, which can be part of integral feasible solutions. Thus, we manipulate the set of feasible solutions. Since we take care that at least one optimal feasible solution remains, the reduction scheme is valid.

Set Dominated Breaks

The detection of irrelevant breaks requires nonnegative energy prices in subintervals of the considered time window. However, if there exists one period $t \in [T[_{\mathbb{Z}}$ with $P_t < 0$, we can still try to detect and eliminate the redundant breaks using a similar approach. In the presence of negative energy prices, the redundancy of break variables is checked by an additional optimization problem since it is not obvious if the negative energy price can reward additional ramping.

Theorem 4.1.41. *Let $m \in M$ be one machine. If the break $(t_0, t_1) \in B_m$ satisfies the condition*

$$\hat{d}_{m,t_0,t_1} \geq \min \Big\{ \sum_{\substack{(q_0,q_1)\in B_m: \\ t_0 \leq q_0 < q_1 \leq t_1}} \hat{d}_{m,q_0,q_1} z^{rd,ru}_{m,q_0,q_1} + \sum_{q=q_0}^{q_1} \hat{d}^{st}_{m,q} z^{st}_{m,q} \;\; | \tag{4.25}$$
$$\sum_{\substack{(q_0,q_1)\in B_m: \\ t\in\{q_0,\dots,q_1\}}} z^{rd,ru}_{m,q_0,q_1} = 1 - z^{st}_{m,t} \quad t \in [t_0, t_1[_{\mathbb{Z}} , \tag{4.26}$$
$$z^{rd,ru}_{m,t_0,t_1} = 0, \tag{4.27}$$
$$z^{rd,ru}_{m,q_0,q_1} \in \{0,1\} \quad \forall (q_0, q_1) \in B_m \cap [t_0, t_1[_{\mathbb{Z}} \times [t_0, t_1[_{\mathbb{Z}} , \tag{4.28}$$
$$z^{st}_{m,t} \in \{0,1\} \quad \forall t \in [t_0, t_1[_{\mathbb{Z}} \Big\}, \tag{4.29}$$

then the break (t_0, t_1) *can be eliminated.*

Proof. Let $(z^{st}, z^{rd,ru})$ be an optimal solution for Problem (4.25)–(4.29). The solution describes an optimal assignment of breaks and standby to the periods $t \in [t_0, t_1[_{\mathbb{Z}}$, without using the break (t_0, t_1) (and neglecting the processing of the task). If the solutions' objective value satisfies

$$\hat{d}_{m,t_0,t_1} \geq \sum_{\substack{(q_0,q_1)\in B_m: \\ t_0 \leq q_0 < q_1 \leq t_1}} \hat{d}_{m,q_0,q_1} z^{rd,ru}_{m,q_0,q_1} + \sum_{q=q_0}^{q_1} \hat{d}^{st}_{m,q} z^{st}_{m,q},$$

the break variable $z^{rd,ru}_{m,t_0,t_1}$ can always be replaced by a combination of the standby in the periods $\{t \in [t_0, t_1[_{\mathbb{Z}} \mid z^{st}_{m,t} > 0\}$ and the breaks $\{(q_0, q_1) \in B_m : z^{rd,ru}_{m,q_0,q_1} > 0\}$. The elimination of (t_0, t_1) does not cut off the optimal solution since the combination has at most the same objective value. Thus, there always exists one feasible solution, which has an equally good or better objective value that does not use the break variable $z^{rd,ru}_{ma,t_0,t_1}$. Thus, the break (t_0, t_1) is redundant and the associated break variable $z^{rd,ru}_{ma,t_0,t_1}$ can be fixed to zero. □

The Problem (4.25)–(4.29) must be solved for each break. The constraint matrix is totally unimodular, and one can obtain a feasible integer solution by the solution of its LP-relaxation. Although we only need to solve the LP-relaxation for nearly every break $(t_0, t_1) \in B_m$, the complete solution time for many LP-relaxations is expensive and a more efficient way to solve the presolving problem (4.25)–(4.29) need to be discussed.

Exploiting the Shortest Path Structure

The assignment problem (4.25)–(4.29) has an underlying structure. We exploit this structure to solve the presolving and propagation problem for each break (t_0, t_1) in a more efficient way.

Example 4.1.42. *Within this example, we want to present the shortest path structure of detecting an assignment of a time window to standby or breaks. Therefore, we only use breaks of equal length since the ramping length does not affect the problem structure.*
To cover the periods $[t_0, t_4[_{\mathbb{Z}}$ *with standby or breaks, the following constellations are possible:*

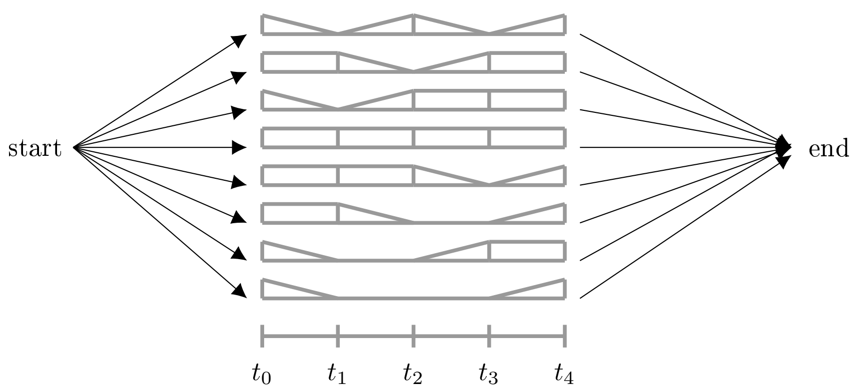


Figure 4.5: Example of all possible choices to cover periods $[t_0, t_1[_{\mathbb{Z}}$ by standby or breaks. The rectangles describe the standby periods, and the triangles ramp-down and ramp-up blocks. A line describes offline periods.

Each possible assignment of the periods $[t_0, t_1[_{\mathbb{Z}}$ *to breaks or standby is visualized by one row in Figure 4.5.*

The following Figure 4.6 uses only the machine states standby *and* offline. *The machine states* ramping-down *and* ramping-up *are visualized by arcs from* offline *to* standby, *respectively the end-node, or* standby, *respectively the start-node to* offline. *The network looks as follows:*

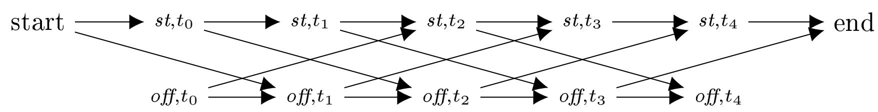


Figure 4.6: This figure shows the acyclic network representation, where no additional data must be stored at each node. The number of nodes is decreased, but the number of arcs increases.

The formal definition of the network, presented in Figure 4.6, is as follows.

Definition 4.1.43 (Network for presolving of one single break)**.** *Let $m \in M$ be one machine and $(t_0, t_1) \in B_m$ one break. The network of the interval $[t_0, t_1[_{\mathbb{Z}}$ is defined by $N^{t_0,t_1} = (D = (V, A), l)$ with*

$$\begin{aligned} V = \{(start), (end)\} &\cup \{(st, t) \mid \forall\, t \in [t_0, t_1[_{\mathbb{Z}}\ \} \\ &\cup \{(off, t) \mid \forall\, t \in [t_0, t_1[_{\mathbb{Z}}\ \} \end{aligned}$$

and

$$\begin{aligned} A = &\cup \big\{ \big((st, t), (st, t+1)\big) \mid\ t, t+1 \in [t_0, t_1[_{\mathbb{Z}}\ \big\} \\ &\cup \big\{ \big((off, t), (off, t+1)\big) \mid\ t, t+1 \in [t_0, t_1[_{\mathbb{Z}}\ \big\} \\ &\cup \big\{ \big((off, t), (st, t + d^{ru}_m)\big) \mid t, t + d^{ru}_m \in [t_0, t_1[_{\mathbb{Z}}\ \big\} \\ &\cup \big\{ \big((st, t), (off, t + d^{rd}_m)\big) \mid t, t + d^{rd}_m \in [t_0, t_1[_{\mathbb{Z}}\ \big\} \\ &\cup \big\{ \big((start), (off, t_0 + d^{rd}_m)\big) \big\} \\ &\cup \big\{ \big((off, t_1 - d^{ru}_m), (end)\big), \big((st, t_1 - 1), end\big) \big\} \\ &\cup \big\{ \big((start), (st, t_0)\big), \big((start), (off, t_0 + d^{rd}_m)\big) \big\} \end{aligned}$$

and the arc lengths $l \in \mathbb{R}^A$

$$\begin{aligned} l_{(start),(st,t_0)} &= P_{t_0} \cdot D^{st}_m \\ l_{((st,t_1-1),(end)} &= 0 \\ l_{(off,t),(off,t+1)} &= 0 && \forall\, t, t+1 \in [t_0, t_1[_{\mathbb{Z}} \\ l_{(st,t),(st,t+1)} &= P_{t+1} \cdot D^{st}_m && \forall\, t, t+1 \in [t_0, t_1[_{\mathbb{Z}} \\ l_{(st,t),(off,t+d^{rd}_m)} &= \sum_{q=t}^{t+d^{rd}_m-1} P_q \cdot D^{rd}_m && \forall\, t, t + d^{rd}_m \in [t_0, t_1[_{\mathbb{Z}} \\ l_{(off,t-d^{ru}_m),(st,t)} &= \sum_{q=t-d^{ru}_m}^{t-1} P_q \cdot D^{ru}_m && t, t + d^{rd}_m \in [t_0, t_1[_{\mathbb{Z}} \\ l_{(start,t+d^{rd}_m)} &= \sum_{q=t_0}^{t_0+d^{rd}_m-1} P_q \cdot D^{rd}_m \\ l_{(off,t_1-1-d^{ru}_m),(end)} &= \sum_{q=t_1-1-d^{ru}_m}^{t_1-1} P_q \cdot D^{ru}_m. \end{aligned}$$

Proposition 4.1.44. *Let (t_0, t_1) be one break belonging to machine $m \in M$. Additionally, let $N^{t_0,t_1} = (D = (V, A), l)$ be a network of the form 4.1.43. Then, one can decide in $\mathcal{O}(|A|)$ whether there exists one optimal solution of the scheduling problem satisfying $z^{rd,ru}_{m,t_0,t_1} = 0$.*

Proof. Let $m \in M$ be a machine and $(t_0, t_1) \in B_m$ one break. The network $N = (D, l)$ describes all possible paths from $(start)$ to (end).
Let P be a $(start)$-(end) path in D. Then, in each period $t \in [t_0, t_1[_{\mathbb{Z}}$, the machine is assigned to one machine state *standby*, *offline* or on an arc, describing the ramping. The arcs from *off* to *st* and from *st* to *off* were set correctly, such that the ramping durations are considered. Each paths from $(start)$ to (end) describes an assignment of all periods $t \in [t_0, t_1[_{\mathbb{Z}}$ to standby and breaks. For an optimal solution of (4.25)–(4.29), only those assignments of periods to machine states are of interest if the assignment has the lowest objective costs. If the break (t_0, t_1) corresponds to a $(start)$-(end) path that does not describe one of the shortest paths in D, then the break will not be used in an optimal solution, since there exists a $(start)$-(end) path, and thus an assignment of the periods to the machine states $s \in \{off, rd, ru, st\}$, such that the objective decreases by switching form break (t_0, t_1) to the less expensive $(start)$-(end) path.

Suppose there are multiple optimal solutions with objective equal to $\hat{d}_{m,t_0,t_1}$. Then, we can detect one path, unequal to the path, corresponding to (t_0, t_1). One possibility is the computation of one shortest path in N with at least one visit of a *st*-node in $[t_0, t_1[_{\mathbb{Z}}$. This could be done by additionally tracking the number of visited *st*-nodes and by only updating the shortest path at node (end) if there was a visit of a *st* node. ☐

Since the network is acyclic, we can use topological sorting to derive the optimal solution in $\mathcal{O}(|A|) = \mathcal{O}(T)$. This presolving and propagation algorithm is only effective before the root node. Since we do not consider local time windows of tasks, the approach cannot detect further reductions within the branch-and-bound tree.

Theorem 4.1.45. *Let $m \in M$ be one machine and $(t_1, t_2), (t_0, t_3) \in B_m$ pairwise distinct breaks satisfying*

$$t_0 \leq t_1 < t_2 \leq t_3.$$

If the break (t_1, t_2) is part of the shortest $(start) - (end)$ path in $N^{t_0,t_3} = (D, l)$, then there exists at least one optimal solution $(x, z^{st}, z^{rd,ru}) \in \mathcal{P}^B$ satisfying $z^{rd,ru}_{m,t_0,t_3} = 0$.

Proof. Let $m \in M$ be one machine and $(t_0, t_3), (t_1, t_2) \in B_m$ pairwise distinct breaks satisfying

$$t_0 \leq t_1 < t_2 \leq t_3.$$

Additionally let (t_1, t_2) be part of one shortest $(start) - (end)$ path in $N^{t_0,t_3} = (D, l)$. The shortest path consists of a set of breaks S^{break} and a set of standby periods S^{st}. Suppose

$$\hat{d}_{m,t_0,t_3} < \sum_{(q_0,q_1) \in S^{break}} \hat{d}_{m,q_0,q_1} + \sum_{q \in S^{st}} P_t \cdot D^{st}_m.$$

Then, the shortest $start) - (end)$ path would use (t_0, t_3) instead of the breaks $(q_0, q_1) \in S^{break}$ and the standby periods $q \in S^{st}$. Thus, this is a contraction to the optimality of the shortest $(start) - (end)$path, including (t_1, t_2). Thus, there exists one optimal solution $(x, z^{st}, z^{rd,ru}) \in \mathcal{P}^B$ satisfying $z^{rd,ru}_{m,t_0,t_3} = 0$. ☐

Corollary 4.1.46. *Let $m \in M$ be one machine and $(t_1, t_2), (t_3, t_4), (t_0, t_5) \in B_m$ pairwise distinct breaks satisfying*

$$t_0 \leq t_3 \leq t_1 < t_2 \leq t_4 \leq t_5.$$

If the break (t_1, t_2) is used by one shortest $(start) - (end)$ path in $N^{t_0,t_3} = (D, l)$, then there exists at least one optimal solution $(x, z^{st}, z^{rd,ru}) \in \mathcal{P}^B$ satisfying $z^{rd,ru}_{m,t_3,t_4} = 0$.

The Corollary 4.1.46 shows that the presolving scheme need not be applied for each break explicitly. Many reductions can be found when computing one single shortest path.

Remark 4.1.47. *The presolving rule 4.1.41 does not guarantee unique optimal solutions since shifting the schedule could lead to feasible solutions with the same objective value. This presolving rule permits feasible solutions, which differ only by substituting breaks with a combination of breaks and standby. However, there could still be distinct optimal solutions using different breaks and the same processing starts.*

Infeasibilities by Combinatorial Probing

The fixations of breaks are not discussed in Scheme (4.14), since these fixations are combined with a more general approach, which generalizes the scheme (4.14) in case of breaks and detects conflicts of subsets of tasks and the usage of one specific break on the same machine. Therefore, we consider a break $(t_0, t_1) \in B_m$ and we analyze the problem on a single-machine after fixing the usage of (t_0, t_1). Possible approaches to detect infeasibilities are proposed. To motivate this combinatorial probing scheme, let's have a look at the following example.

Example 4.1.48. *We are given three tasks $O^M_{|m} = \{(1,0), (2,0), (3,0)\}$ on machine m. The setup and processing durations of the tasks are 2 for each task. The time windows of each task $(j,k) \in \{(1,0), (2,0), (3,0)\}$ is set to $a_{j,k} = 6$ and $f_{j,k} = 34$. The ramping duration of machine m is $d^{rd}_m = 2$ and $d^{ru}_m = 4$. One wants to know if break $(t_0, t_1) = (6, 26)$ can be used within an optimal integral solution. Fractionally, the break $(6, 26)$ can be used in combination while processing the tasks $(1,0), (2,0)$ and $(3,0)$, since the condition*

$$t_1 - t_0 + \sum_{(j,k) \in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k}) = 26 - 6 + 12 \leq 36 - 4 = 32 \tag{4.30}$$

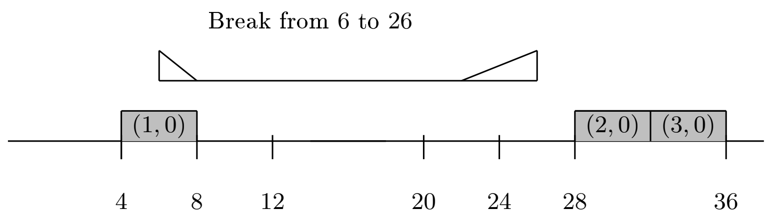


Figure 4.7: Illustration the tasks $(1, 0), (2, 0), (3, 0)$ and the conflict of using the break $(6, 26)$ with the release and due date $a_m = 6$ and $f_m = 36$.

holds. Thus, the bound on the length of middle breaks would not detect a possible reduction. However, the break $(6, 26)$ *should not be used even in a feasible solution. The break* $(6, 26)$ *cannot be moved, and the task* $(1, 0)$ *cannot start earlier, but the processing of* $(1, 0)$ *and the break overlap. Thus, the processing of* $(1, 0)$ *cannot start in period* 6 *while using break* $(6, 26)$*. Therefore, one should shift the task* $(1, 0)$ *onto the right side of break* $(6, 26)$*. Then, the processing can start in period* 28*. However, there are only* 10 *periods left to complete the processing of three tasks with a processing and setting-up duration of* 12*. Thus, the fixed usage of break* $(6, 26)$ *leads to infeasibility and the break cannot be used in feasible solutions.*

Now, we consider one single-machine $m \in M$ and the tasks $(j, k) \in O^M_{|m}$. The break (t_0, t_1) cannot be used in combination with the local time windows $a_{j,k}$ and $f_{j,k}$ of each task $(j, k) \in O^M_{|m}$ if the following integer program does not have any integral solutions:

$$\text{minimize} \quad 0 \tag{4.31a}$$

subject to

$$\sum_{t \in [l,r[_{\mathbb{Z}}} x_{j,k,t} = 1 \qquad \forall (j,k) \in O^M_{|m} \tag{4.31b}$$

$$z^{rd,ru}_{m,q_0,q_1} = 1 \tag{4.31c}$$

$$\sum_{(j,k) \in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t+d^{se}_{j,k}} x_{j,k,q} \leq 1, \qquad t \in [T[_{\mathbb{Z}} \tag{4.31d}$$

$$\sum_{(j,k) \in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t+d^{se}_{j,k}} x_{j,k,q} \leq 0, \qquad t \in [q_0, q_1[_{\mathbb{Z}} \tag{4.31e}$$

$$x_{j,k,t} \in \{0, 1\}, \quad (j,k) \in O^M_{|m},\ t \in [l, r[_{\mathbb{Z}} \tag{4.31f}$$

$$z^{rd,ru}_{m,t_0,t_1} \in \{0, 1\}, \quad (t_0, t_1) \in B_m. \tag{4.31g}$$

The problem (4.31a)–(4.31g) without constraint (4.31c) is only a relaxation of the formulation (3.10a)–(3.10h). Thus, if (4.31a)–(4.31g) in combination with (4.31c) has no solution, then also (3.10b)–(3.10h) in combination with (4.31c) has no solution.

Problem (4.31a)–(4.31g) can be solved by integer linear programming. However, it is a single-machine scheduling problem with time windows. This problem is known to be NP-hard [KV12], and solving such a hard problem to decide whether a variable is used in a feasible integral solution is excessive. Therefore, this section deals with providing solution strategies to solve the problem efficiently using approximation algorithms.

The fixation of break (t_0, t_1) within the time window splits the set of tasks $O^M_{|m}$ into two sets and the time window also into two sub time windows. Thus, this problem can be considered to be a double knapsack problem or a multi dimensional knapsack problem.

Remark 4.1.49. *The following problem describes a relaxation of* (4.31a)–(4.31g)*.*

Let $m \in M$ *be one machine and* $(t_0, t_1) \in B_m$ *a break. We consider the subset of tasks* $S \subseteq O^M_{|m}$*. The multi dimensional knapsack problem can be built as follows.*

- *Compute* $a_m = \min_{(j,k) \in S} a_{j,k} - d^{se}_{j,k}$ *and* $f_m = \max_{(j,k) \in S} f_{j,k} - 1 + d^{pr}_{j,k}$*.*
- *Fix the variable* $z^{rd,ru}_{m,t_0,t_1} = 1$*.*
- *Create two knapsacks* A *and* B*.*
  - *The knapsack* A *with size* $b_A = \max(t_0 - a_m, 0)$*.*

- *The knapsack B with size $b_B = \max(f_m - t_1, 0)$.*
- *Create the set U consisting of the items $(j,k) \in S$ with size $w_{(j,k)} = d^{pr}_{j_i,k} + d^{se}_{j,k}$ for $(j,k) \in S$.*
- *The item values $c_{(j,k)} = 1$ are chosen equally for each $(j,k) \in S$.*
- *The objective is to maximize the value of the chosen items.*

The described problem is a **multi dimensional knapsack problem** with dimension two. Solving this knapsack problem is equally hard as deciding whether several items fit into two bins. Thus, we can already provide a polynomial time approximation algorithm with an approximation factor of $\frac{3}{2}$ by approximation algorithms for the bin-packing problem. For more insights, see [KV12].

However, we want to devise algorithms that can compute near-optimal solutions. Therefore, we need simplified conditions to verify whether the break can be fixed to zero.

Theorem 4.1.50. *Let $(t_0, t_1) \in B_m$ be a break belonging to machine $m \in M$. The break (t_0, t_1) can be eliminated if there exists a subset $S \subseteq O^M_{|m}$ such that the problem mentioned in Remark 4.1.49 has an optimal solution with objective value smaller than $|S|$.*

Proof. Let $m \in M$ be one machine and $(t_0, t_1) \in B_m$ a break belonging to machine m considered in the combinatorial probing problem. Moreover, let $S \subseteq O^M_{|m}$ be an arbitrary subset of tasks. Suppose the break (t_0, t_1) can still be used in a locally feasible solution of the corresponding single-machine scheduling problem (4.31b)–(4.31g), although the problem mentioned in Remark 4.1.49 has no solution. Let x^* be the local feasible solution of this single-machine scheduling problem of the tasks. Then, the following sets are defined:

$$I_A = \{(j,k) \in S \mid \sum_{t \in [T[_{\mathbb{Z}}} x^*_{j,k,t} \cdot t < t_0\},$$

$$I_B = \{(j,k) \in S \mid \sum_{t \in [T[_{\mathbb{Z}}} x^*_{j,k,t} \cdot t \geq t_1\}.$$

Since x^* is part of a feasible solution of (4.31b)–(4.31g), $I_A \dot\cup I_B = S$ and

$$\sum_{(j,k) \in I_A} d^{pr}_{j_i,k} + d^{se}_{j,k} \leq \max(t_0 - a_m, 0) = b_A,$$

$$\sum_{(j,k) \in I_B} d^{pr}_{j_i,k} + d^{se}_{j,k} \leq \max(f_m - t_1, 0) = b_B$$

hold. The sets I_A and I_B are feasible assignments of the items to the knapsacks A and B. The corresponding objective value is $|I_A \dot\cup I_B| = |S|$. Thus, the considered solution of the problem mentioned in Remark 4.1.49 is not optimal. ☐

Note that the reverse direction does not hold, since multiple constraints and further machines are not considered.

Remark 4.1.51. *The multidimensional knapsack problem is only a relaxation of the considered single-machine scheduling problem. The fixation of a break $(t_0, t_1) \in B_m$ can lead to infeasibility, although the double knapsack problem provides a feasible integer solution with objective value $|O^M_{|m}|$.*

To solve the multidimensional knapsack problem, we use Algorithm 3.

Theorem 4.1.52. *Algorithm 3 computes an optimum solution of the problem described in Remark 4.1.49, and its runtime is bounded by $\mathcal{O}(2^{|O^M_{|m}|})$.*

Proof. The runtime: we create two possible outcomes for each item and restart and evaluate the recursive function with the remaining items. This is done to the depth of at most $n = |O^M_{|m}|$.
The algorithm is a recursion. In stage i, the two possibilities are checked.

1. If item i fits in the left knapsack, then compute the best solution using the items $\{i+1, \ldots, n\}$ and the current remaining capacities.
2. If item i fits in the right knapsack, then compute the best solution using the items $\{i+1, \ldots, n\}$ and the current remaining capacities.

Algorithm 3 recursive Double Knapsack Algorithm **rDKA**(n,b_A,b_B,w,i)

Require: number of items n, knapsacks b_A and b_B and items $(w_i)_{i=1,...,n}$
  if i == n **then**
    return 0
  end if
  $posFill_A = 0$
  $posFill_B = 0$
  if $b_0 \geq w_i$ **then**
    $posFill_A = c_i +$ **rDKA**$(n, b_A - w_i, b_B, i+1, c)$
  end if
  if $b_1 \geq w_i$ **then**
    $posFill_B = c_i +$ **rDKA**$(n, b_A, b_B - w_i, i+1, c)$
  end if

  return $\max(posFill_A, posFill_B)$

If the item i does not fit either in knapsack A or in knapsack B, then there is no need to visit further nodes within this recursion branch.
The recursion enumerates all possibilities and only aborts the search within a branch if the solution we seek cannot be in the current branch. Therefore, the optimal solution will be detected, and the solution time of this algorithm is bounded by $\mathcal{O}(2^{|O^M_{|m}|})$. □

Algorithm 3 performs badly for large sets $S \subseteq O^M_{|m}$. The solution of the double knapsack algorithm is also only a weak relaxation of the associated single-machine scheduling problem. This relaxation currently does not consider the local time windows of tasks. The time windows can permit processing a specific task after or before the break (t_0, t_1). The next presolving steps are supposed to strengthen the relaxation by the double knapsack problem by predetermining the relative position of each task to the break variable. Therefore, we consider the knapsack A to describe the tasks processed before the break (t_0, t_1), and the knapsack B describes the tasks processed after break (t_0, t_1).

Example 4.1.53. *We consider an example similar to the example of Figure 4.7. We reuse the same setting but the task* $(1,0)$ *has the time window* $[6, 21[_{\mathbb{Z}}$ *. Moreover, we consider the break* $(6, 24)$*. Obviously, task* $(1,0)$ *cannot start processing before break* $(6,24)$*. Due to the time window of task* (j,k)*, the task* $(1,0)$ *cannot be processed after break* $(6,24)$*. Thus, we can detect infeasibility by the knowledge of the assignment of the tasks if we consider the time windows.*

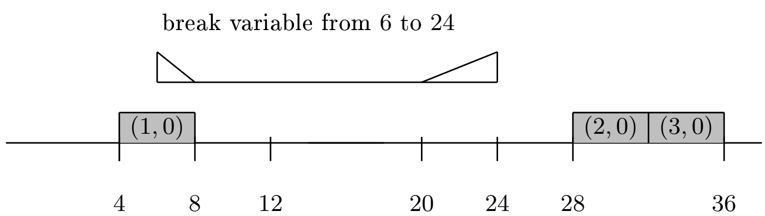


Figure 4.8: Illustration the tasks $(1,0), (2,0), (3,0)$ and the conflict of using the break $(6,24)$ with the release and due date $a_m = 6$ and $f_m = 21$ of task $(1,0)$.

Theorem 4.1.54. *Let* (t_0, t_1) *be one break belonging to machine* $m \in M$*. If the machine* m *is using the break* (t_0, t_1)*, then the task* $(j,k) \in O^M_{|m}$ *can only be processed after period* t_1 *if*

$$a_{j,k} + d^{pr}_{j,k} > t_0$$

holds.

Proof. Let $m \in M$ be one machine and $(t_0, t_1) \in B_m$ one break. The task $(j,k) \in O^M_{|m}$ satisfies the condition $a_{j,k} + d^{pr}_{j,k} > t_0$. Suppose the break starts processing before t_0. Then, the processing of task (j,k) starts at least in period $t_0 - d^{pr}_{j,k}$. Since $a_{j,k} + d^{pr}_{j,k} > t_0$ holds, the processing of task (j,k) starts in $a_{j,k} - 1$, which is not feasible. Thus, the task (j,k) can be fixed to be start processing after break (t_0, t_1). □

Analogously, the following result is valid for tasks that cannot be processed after the break.

Theorem 4.1.55. *Let (t_0, t_1) be one break belonging to machine $m \in M$. If the machine m is using the break (t_0, t_1), then the task $(j,k) \in O^M_{|m}$ can only be processed before period t_0, if*

$$f_{j,k} - d^{se}_{j,k} < t_1$$

holds.

Since we can fix the position of the tasks in relation to the break, we can also fix the assigned item of the multi dimensional knapsack problem to the corresponding knapsack. Using this knowledge of possible item fixations to specific knapsacks leads to the following theorem.

Theorem 4.1.56. *Let $(t_0, t_1) \in B_m$ one break belonging to machine $m \in M$. Additionally, let $S \subseteq O^M_{|m}$ be a subset of tasks. Denote $L \subseteq S$ the subset of tasks that need to be processed before t_0 and $R \subseteq S$ as the set of tasks that need to be processed after t_1. Then, we get new knapsack capacities:*

- $b_A \leftarrow b_A - \sum_{(j,k)\in L} w_{(j,k)}$
- $b_B \leftarrow b_B - \sum_{(j,k)\in R} w_{(j,k)}$

and a new set of unfixed items $U = S \setminus (L \cup R)$. If the corresponding knapsack problem has an optimal solution with objective value $< |U|$, then the break (t_0, t_1) can be eliminated.

Proof. Let $(t_0, t_1) \in B_m$ be the break to be fixed. If $U = S \setminus (L \cup R)$ and $L = R = \emptyset$, the validity of this theorem is proven. If $L \neq \emptyset$, then each item in $(j,k) \in L$ must be part of knapsack A, and the corresponding task (j,k) cannot be processed after break (t_0, t_1). Analogously, the position of the tasks in R to the break (t_0, t_1) are fixed. Therefore, the remaining items in U must be considered within the computation. Those items can be assigned to both knapsacks. If the knapsacks can contain all $|U|$ items, then the objective with consideration of S is $|S|$. Otherwise, one item in U cannot be assigned to A or B, and thus, the objective of the multi dimensional knapsack problem would be smaller than $|U|$. Thus, the multi dimensional knapsack problem has objective $< |S|$. Thus, (t_0, t_1) can be eliminated. □

The double knapsack problem can be solved for each break $(t_0, t_1) \in B_m$. It is preferable to decide by a combinatorial condition whether the double knapsack problem has an objective value of $|S|$. Therefore, a few simple cases do not require the use of the algorithm.

Theorem 4.1.57. *Let $m \in M$ be one machine and $S \subseteq O^M_{|m}$ a subset of tasks. Further, let $(t_0, t_1) \in B_m$ be a break belonging to machine $m \in M$. Denote $L \subset O^M_{|m}$ the set of tasks that need to be processed before t_0 and $R \subset O^M_{|m}$ the set of tasks that need to be processed after t_1. We define:*

- $b_A \leftarrow b_A - \sum_{(j,k)\in L} w_{(j,k)}$
- $b_B \leftarrow b_B - \sum_{(j,k)\in R} w_{(j,k)}$

and $U = S \setminus (L \cup R)$. The break variable $z^{rd,ru}_{m,t_0,t_1}$ cannot be used in a feasible local solution if the condition

$$\sum_{(j,k)\in U} (d^{pr}_{j,k} + d^{se}_{j,k}) > b_A + b_B \tag{4.32}$$

holds.

Proof. The items of size $\sum_{(j,k)\in U}(d^{pr}_{j,k} + d^{se}_{j,k})$ cannot fit into two knapsacks of size $b_A + b_B$. Thus, at least one item $(j,k) \in S$ cannot be assigned to knapsack A and knapsack B. The objective of the corresponding multi dimensional knapsack problem must be smaller than $|S|$. Thus, the break (t_0, t_1) can be fixed to zero. □

This condition describes that the usage of the break variable $z^{rd,ru}_{m,t_0,t_1}$ prevents at least one task from starting processing within the local time windows since at least one item, e.g., one task, cannot be assigned to one side of the break. This condition is a more specific version of Condition (4.13) for the complete set of tasks processed on machine $m \in M$.

However, we prefer to run the algorithm and retrieve the result to fix the variable. Moreover, we do not want to run the algorithm very often without any fixations, since the run of one algorithm can be expensive. Often, the required optimization problem need not be solved. We can compute its result by evaluating combinatorial conditions.

Theorem 4.1.58. *Let $(t_0, t_1) \in B_m$ be a break belonging to machine $m \in M$. Further, let $S \subseteq O^M_{|m}$ be a nonempty subset of tasks. Denote $L \subseteq S$ the set of tasks that need to be processed before t_0 and $R \subseteq S$ the set of tasks that need to be processed after t_1. The new knapsack capacities are*

- $b_A \leftarrow b_A - \sum_{(j,k) \in L} w_{(j,k)}$
- $b_B \leftarrow b_B - \sum_{(j,k) \in R} w_{(j,k)}$

and the new set of unfixed items is $U = S \setminus (L \cup R)$. The multi dimensional knapsack problem has objective $|S|$, if the condition

$$b_A + b_B - \sum_{(j,k) \in S} (d^{pr}_{j,k} + d^{se}_{j,k}) \geq \max_{(j,k) \in U} (d^{pr}_{j,k} + d^{se}_{j,k}) \tag{4.33}$$

is satisfied.

Proof. Let $m \in M$ be one machine and $(t_0, t_1) \in B_m$ a break belonging to machine m. The break (t_0, t_1) only can be used locally, if the condition

$$\sum_{(j,k) \in U} (d^{pr}_{j,k} + d^{se}_{j,k}) \leq b_A + b_B$$

holds. Suppose the item (i, l) cannot be assigned to knapsack A and knapsack B. The double knapsack algorithm fits the items without gaps. Denote I_A the items assigned to knapsack A and I_B the items assigned to knapsack B. Thus, the remaining capacity in both knapsacks satisfies

$$\hat{b}_A = b_A - \sum_{(j,k) \in I_A} (d^{pr}_{j,k} + d^{se}_{j,k}) < d^{pr}_{i,l} + d^{se}_{i,l}$$

and

$$\hat{b}_B = b_B - \sum_{(j,k) \in I_B} (d^{pr}_{j,k} + d^{se}_{j,k}) < d^{pr}_{i,l} + d^{se}_{i,l}.$$

Because $\sum_{(j,k) \in U} (d^{pr}_{j,k} + d^{se}_{j,k}) \leq b_A + b_B$ holds, the reduced capacities satisfy $\hat{b}_A + \hat{b}_B > d^{pr}_{i,l} + d^{se}_{i,l}$. Thus $\hat{b}_A > \frac{d^{pr}_{i,l} + d^{se}_{i,l}}{2}$ or $\hat{b}_B > \frac{d^{pr}_{i,l} + d^{se}_{i,l}}{2}$ holds. To be able to assign the item (j, k) to one of the knapsacks, we enlarge both knapsacks by $\frac{d^{pr}_{i,l} + d^{se}_{i,l}}{2}$. Then, at least one of the knapsacks $i \in \{A, B\}$ has a reduced size of $\hat{b}_i \geq d^{pr}_{i,l} + d^{se}_{i,l}$. Thus, to be able to consider each possible task, the global additional capacity is $\max_{(j,k) \in U} (d^{pr}_{j,k} + d^{se}_{j,k})$.

Thus, if the originally multi dimensional knapsack problem satisfies

$$b_A + b_B - \sum_{(j,k) \in S} (d^{pr}_{j,k} + d^{se}_{j,k}) \geq \max_{(j,k) \in U} (d^{pr}_{j,k} + d^{se}_{j,k}),$$

then the multi dimensional knapsack problem has objective $|S|$. □

This condition follows from the $\frac{3}{2}$ approximation algorithm of the bin packing approximation. There is the possibility to apply the presolving and propagation rule (4.17) to the tasks fixed to L respectively R. The tasks of the set U must be ignored.

This reduction scheme can be applied within the branch-and-bound tree. Initially, the number of reductions by this presolving scheme is low since this presolving scheme requires tight time windows for all tasks. To apply this presolving, the branch-and-bound algorithm needs to shrink the time windows of the tasks.

4.2 The Branch-and-Bound Algorithm

Integer linear programs are commonly solved by enumerative methods, called branch-and-bound and branch-and-cut, see for example [LD60, AKM05, WN14]. In this section, we present the basic principles of branch-and-bound and touch on the traditional branching techniques employed by commercial MILP solvers. After that, we discuss known branching rules for scheduling and their effect on the job-shop scheduling problem with flexible energy prices and time windows. Then, we introduce our problem-specific branching rules. Through analysis of a specific instance's LP relaxation, we demonstrate the necessity for problem-specific branching rules. We are using constraint-based branching to enforce the integrality of the workload of the machines, and we use constraint-based branching to shrink the time windows of the tasks. At the same time, various variants of the branching candidate selection rules are presented with a discussion of their advantages and disadvantages.

4.2.1 Branch-and-Bound in General

A well-known method for solving integer programs is to use a branch-and-bound algorithm ([LW66, CCZ14]) in combination with linear programming to derive the bounds. Given an integer program

$$\min\{c^\top x \mid \; Ax \leq b, \; x \in \mathbb{Z}^n\} \tag{4.34}$$

with $A \in \mathbb{Q}^{m\times n}$, $c \in \mathbb{Q}^n$ and $b \in \mathbb{Q}^m$. The algorithms start by initializing the list of open problems, called **open branch-and-bound nodes**, with the original problem. The idea of the branch-and-bound algorithm is to recursively divide the problems into disjunctive subproblems to improve the best-known lower bound towards the optimal objective value if there exists one feasible solution until the list of open nodes is empty. The smallest lower bound of the open nodes is called the **dual bound**. The best-known feasible solution's objective is the **primal bound**. One node of the list of open nodes is selected and solved, for example, by linear programming, to provide a lower bound to the optimal objective value of the selected node.

If the solution values x^* of the LP-relaxation of a branch-and-bound node of (4.34) are integral, and the corresponding objective value improves the primal bound, the primal bound objective value is updated. If the current problem is infeasible, further branching cannot repair the infeasibility, and the node has no feasible solution. If the computed bound exceeds the primal bound, this local subtree cannot contain a feasible integer solution, improving the current primal bound. If the solution x^* of (4.34) is fractional, a branching creates two new problems, called **child nodes**. The child nodes are created by introducing a **branching decision**, which can be expressed by two linear constraints dividing the problem into these disjunctive subproblems. Next, the branch-and-bound tree, and thus the list of open nodes, is expanded by adding two child nodes and the next node to solve is chosen by a rule called **node selection**.

A well-known strategy to divide the problem into smaller subproblems is variable branching. If the solution x^* of (4.34) is fractional, a fractional variable $x_i \notin \mathbb{Z}$, $i \in \{1, \ldots, n\}$, exists. The space of feasible solutions is partitioned into multiple subsets. In the classical branch-and-bound algorithm, the variable branching constraints: $x_i \leq \lfloor x_i^* \rfloor$ and $x_i \geq \lceil x_i^* \rceil$ are used to realize the branching.

The branch-and-bound algorithm terminates if the list of open nodes is empty or the primal bound equals the dual bound. Some well-known variable branching rules, for example, strong branching or pseudo-cost branching, are explained in [AKM05].

General Disjunctions and Branching on Constraints

A branch-and-bound algorithm is not limited to branch on variables. Moreover, general disjunctions can be used in the branch, and bound algorithm [NCKL11, MJSS16, TBBK23] to speed up the solution process. In the case of the usage of linear constraints, the branching takes the form of:

$$\begin{aligned} \text{child A:} \quad & \sum_{i\in J} a_i x_i \leq m \\ \text{and} \quad & \\ \text{child B:} \quad & \sum_{i\in J} a_i x_i \geq m+1 \end{aligned}$$

with $a \in \mathbb{Z}^n$, $m \in \mathbb{Z}$ and $\emptyset \neq J \subseteq \{1, \ldots, n\}$. For $J = \{j\}$ and $m = \lfloor \overline{x}_j \rfloor$, variable branching can be realized. This way of dividing the solution space generalizes the classical variable branching. The branching will change the fractional solution in both child nodes if the condition

$$m < \sum_{i\in J} a_i x_i < m+1$$

holds. Otherwise, the fractional solution remains feasible for at least one subproblem, and thus, the same branch could be created within the respective subproblem again, and finally, the branch-and-bound algorithm will not terminate.

In the case of set-packing problems, early steps in the field of constraint branching were taken in the publications [RF81, Etc77]. In both publications, the authors remark on the possibility of a bad performance of the variable branching, resulting from the unbalanced strong impact on the child nodes.

The classical variable branching results in unbalanced strong branches [RF81][p.279]. One branch significantly improves the solving process, while the other branch has (nearly) no impact on the dual bound. In addition to the consideration of more general branching conditions, the authors propose matrix property-preserving branching constraints.

As an example, we consider the **Ryan-Foster branching** for $A \in \{0,1\}^{m \times n}$. Consider the two different rows $A_{i,:}x \leq b_i$ and $A_{j,:}x \leq b_j$ of the system, with $i, j \in \{1, \dots, m\}$. Then,

$$\sum_{k=1}^{n} A_{i,k} \cdot A_{j,k} x_k = \sum_{k \in \{1,\dots,m\} : A_{i,k} > 0 \text{ and } A_{j,k} > 0} x_k$$

holds. If the LP relaxations' solution of the current node is fractional, then there exists at least two distinct rows $i, j \in \{1, \dots, m\}$ with $0 < A_{r,:}x < 1$ for $r = i, j$. Then, the constraint $\sum_{k=1}^{n} A_{i,k} \cdot A_{j,k} x_i \geq 1$ and constraint $\sum_{k=1}^{n} A_{i,k} \cdot A_{j,k} x_i \leq 0$ realize a valid branching. The branch $\sum_{k=1}^{n} A_{i,k} \cdot A_{j,k} x_i \geq 1$ branch forces the solution to cover the constraints i and j with the same variable. The $\sum_{k=1}^{n} A_{i,k} \cdot A_{j,k} x_i \leq 0$ branch forces to cover the constraints i, j by different variables.

The experiments in [RF81] verified their assumption that the resulting branches become similarly strong by using this branching. In the literature and the associated experiments, this branching was applied successfully, and the generation of more balanced branches was observed multiple times.

Another well-known example of branching on constraints is the so-called **SOS branching** [BT69, FP17]. This problem-specific branching rule was developed for *special ordered set* (SOS). Beale and Tomlin introduced the SOS branching in [BT69]. A certain kind of inequality characterizes a special ordered set:

- SOS1 constraint: at most, one variable from the set can take a non-zero value.
- SOS2 constraint: at most, two variables out of the set are allowed to take a non-zero value, and the two must be adjacent variables.

To explain the basic idea of SOS branching, we are using the SOS1 constraint

$$\sum_{r=1}^{n} x_r = 1, \tag{4.35}$$

with binary variables $x \in \{0,1\}^n$. For $k \in [n-1[_{\mathbb{Z}}$, let x_k and x_{k+1} be a pair of variables in constraint (4.35). Since only one variable of the set $\{x_1, \dots, x_n\}$ can take a non-zero value, the non-zero value can be found beyond x_k or before x_{k+1}.

The introduction of the branching-constraints

$$\text{child A} \quad \sum_{r=1}^{k} x_r = 1, \tag{4.36}$$

and

$$\text{child B} \quad \sum_{r=k+1}^{n} x_r = 1, \tag{4.37}$$

lead to two different subproblems, if $0 < \sum_{r=k+1}^{n} x_r < 1$ holds. The selection of the suitable index k remains, and the choice will differ depending on the application. Different choices are considered in [BT69, FP17]

Van den Akker took up the idea of SOS branching. She successfully applied the idea of those disjunctions in the context of time-indexed formulation in scheduling in [vdA94]. The branching rules that we have developed for the scheduling problem with flexible energy prices and time windows also fall into the category of SOS branching.

4.2.2 Challenges in Fractional Solutions

The fractional solutions of the job-shop scheduling problem with flexible energy prices and time windows need to be treated differently than the fractional solutions of classical scheduling problems with objective makespan or weighted completion time. In the following section, we discuss the use of an often-used branching rule for scheduling, namely the fixation of precedences.

But before analyzing, we first define the concept of workload in order to be able to talk about the properties of fractional solutions

Definition 4.2.1. *For machine $m \in M$ and period $t \in [T[_{\mathbb{Z}}$, the activity or the workload of m in t is defined by*

$$w_{m,t} := \sum_{(j,k) \in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t+d^{se}_{j,k}} x_{j,k,q} + z^{st}_{m,t}.$$

Precedence Constraints Branching

Branching and disjunctions are not limited to one single linear constraint per node. Additionally, a new set of valid inequalities can describe the disjunction of the branching decision at a branch-and-bound node. Nevertheless, it is crucial to note that the branching still represents a disjunction of the previous solution space. An instance of this is the precedence constraints branching.

When scheduling tasks on machines, the main focus is typically determining one optimal execution order of the tasks. The problem formulation of this ordering problem was previously outlined in Section 3.2.4. This problem extension uses the ordering variables $p_{j,k}^{i,l}$ for all pairs of pairwise distinct tasks $(j,k),(i,l) \in O_{|m}^{M}$. The extension requires $\mathcal{O}(T \cdot |O|)$ many constraints linking the ordering variables and the task variables. By using a classical variable branching on the $p_{j,k}^{i,l}$ variables, branching on the execution order of the variables can be realized. However, $\mathcal{O}(T \cdot |O|)$ many precedence constraints are required initially.

The amount of work required to manage the redundant inequalities is too large. After each branching, at most T-many constraints of the linear ordering formulation and the coupling unfixed precedence constraints become active while T-many constraints become redundant at each node. Therefore, we are not interested in initially adding all those constraints to the formulation.

A more memory-saving variant is the adaptive extension of the problem formulation by precedence constraints as branching decisions at each branch-and-bound node. The so-called disjunctive graph [Pin08, p.179] is a way of visualizing the fixed and unfixed precedence relations.

Definition 4.2.2. *For a set of machines M and a set of tasks O, mapped to the machines by $M : O \to M$, a graph $G = (V, C \dot\cup D)$ is called* ***disjunctive graph*** *if*

1. *V is the set of task O.*
2. *$C := \Big\{\big((j,k),(j,k+1)\big) \mid \forall\ (j,k),(j,k+1) \in O_{|j}^{J},\ \forall\ j \in J\Big\}$ is the set of all fixed precedences of job-sequences .*
3. *$D := \big\{\big((j,k),(i,l)\big) \mid (j,k),(i,l) \in O_{|m}^{M},\ (j,k) \neq (i,l)\ \forall\ m \in M\big\}$ is the set of unfixed precedences of tasks, being processed by the same machine.*

Tasks $(j,k),(j,l) \in O$, which are not linked in the graph G, can be processed in parallel on different machines.

Pinedo presents the disjunctive graph in [Pin08] in combination with disjunctive programming to fix precedence relations. In that context, the definitions of conjunction and disjunction are important.

Definition 4.2.3. *A set C of constraints is called conjunctive if each constraint $c \in C$ must be satisfied. A set D of constraints is called disjunctive if at least one constraint $d \in D$ must be satisfied.*

An example of a conjunction is the precedence order of the tasks $(j,k),(j,k+1) \in O_{|j}^{J}$. An example of a disjunction is the information that either (j,k) precedes task (i,l) or (i,l) precedes task $(j,k$, for two distinct tasks $(j,k),(i,l) \in O_{|m}^{M}$.

Not all disjunctions $d \in D$ are of interest to branch on. The local time windows of each task are used to decide whether the precedence constraints between two tasks $(j,k),(i,l) \in O_{|m}^{M}$ one machine m are valid and necessary. An indicator of whether the time windows of the tasks $(j,k),(i,l)$ allow the processing of the tasks in an arbitrary order is the flexibility of the tasks, among others, mentioned in [Pin08].

Definition 4.2.4 (Flexibility of two tasks by Pinedo [Pin08])**.** *Let $(j,k),(i,l) \in O_{|m}^{M}$ two different tasks on machine $m \in M$. The flexibility of the tasks (j,k) and (i,l) is defined by*

$$flex_{j,k,i,l} = \operatorname{sgn}(\sigma_{(j,k)\to(i,l)}) + \operatorname{sgn}(\sigma_{(i,l)\to(j,k)}) \tag{4.38}$$

with

$$\sigma_{(j,k)\to(i,l)} = f_{i,l} + d_{i,l}^{pr} - 1 - a_{j,k} - d_{j,k}^{se} - d_{i,l}^{pr}.$$

If the flexibility is positive, then the time windows of the tasks allow an arbitrary ordering of the tasks (j,k) and (i,l). If the flexibility is zero, then there is an implicit ordering of the tasks (j,k) and (i,l). Finally, if the flexibility is negative, the local time windows do not allow the completion of both tasks within their local time windows, and the current node is infeasible.

If there are two tasks $(j,k),(i,l) \in O^M_{|m}$ processed by machine $m \in M$ satisfying $flex_{j,k,i,l} > 0$, then branching on the corresponding precedence constraints could be useful.

However, the branching is only meaningful if the current LP relaxation is not feasible in both child nodes. Thus, the current LP relaxation must violate each precedence constraint, interpreted as a conjunction. The following theorem describes a precedence constraint branching scheme.

Proposition 4.2.5. *Choosing the tasks $(j,k),(i,l) \in O^M_{|m}$ with $flex_{j,k,i,l} > 0$ on $m \in M$ by the following optimization problem*

$$\max_{t_1,t_2 \in [T[_{\mathbb{Z}}} \Big(\big(\sum_{q=0}^{t_1 - d^{pr}_{i,l} - d^{se}_{j,k}} x_{i,l,q} - \sum_{q=0}^{t_1} x_{j,k,q} \big) \cdot \big(\sum_{q=0}^{t_2 - d^{pr}_{j,k} - d^{se}_{i,l}} x_{j,k,q} - \sum_{q=0}^{t_2} x_{i,l,q} \big) \Big) > 0$$

leads to a valid branching.

The branching is realized by creating two child nodes A and B of the current branch-and-bound node and extending the problem formulation with the following constraints:

$$\text{node A} \qquad \sum_{q=0}^{t - d^{pr}_{j,k} - d^{se}_{i,l}} x_{j,k,q} - \sum_{q=0}^{t} x_{i,l,q} \geq 0 \quad \forall\, t \in [T[_{\mathbb{Z}}$$

$$\text{node B} \qquad \sum_{q=0}^{t - d^{pr}_{i,l} - d^{se}_{j,k}} x_{i,l,q} - \sum_{q=0}^{t} x_{j,k,q} \geq 0 \quad \forall\, t \in [T[_{\mathbb{Z}} \,.$$

This branching rule helps to find near-optimal solutions since the order of the tasks is defined quickly. However, this branching rule is unsuitable for driving the dual bound in the case of the job-shop scheduling problem with flexible energy prices and time windows.

Example 4.2.6. *We consider the instance **Dataorig_ver1_1**. The instance is presented in Section A.2.1. If we solve the LP-relaxation of this instance, the machine's profile on machine $m = 2$ looks as follows: Figure 4.9 shows the fractional workload of machine $m =$*

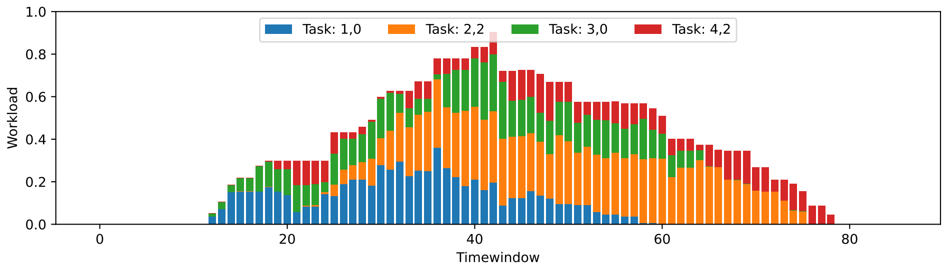


Figure 4.9: Solving the instance with the necessary precedence constraints on machine $m = 2$. The x-axis describes the time window and the periods and the y-axis is the utilization of the machine.

2. One can guess that there is an order of the tasks processed on machine $m = 2$. However, the optimal order is blue - green - orange - red. The fixation of the order does not fix the problem of non-integral workload. The processing of the tasks is still overlapping as long as the processing starts are not fixed, and additional branching is necessary. Figure 4.10

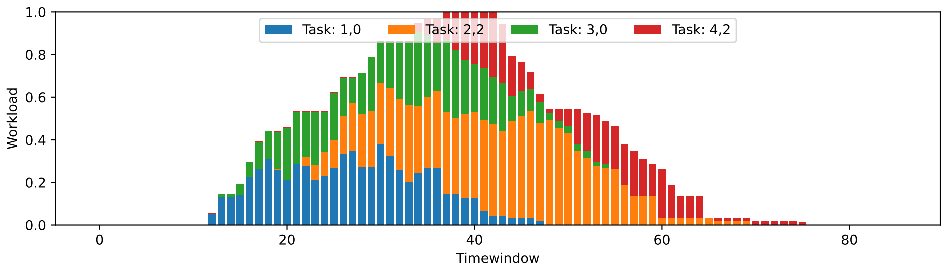


Figure 4.10: Solving the instance with a fixed optimal execution order. The x-axis describes the time window and the periods and the y-axis is the utilization of the machine.

shows that the information about the optimal execution order is insufficient to describe the optimal integer feasible solution.

Extending the ILP formulation by additional precedence constraints sharpens the description of the optimal solution. The solution is still not integral. Thus, the complete information about the precedence order does not lead to integral solutions in the case of the job-shop scheduling problem with flexible energy prices in general. Thus, branching on precedence constraints does not lead to optimal solutions, and more branching rules are required.

Classical Variable Branching

Variable branching is a well-known branching technique. As mentioned before, variable branching can become inefficient, for example, in the case of set packing formulations, since the resulting branches are unbalanced. The unbalanced branches can lead to large branch-and-bound trees, and the exploration of a large number of branch-and-bound nodes is time-consuming. Van den Akker confirms those results in her thesis [vdA94]. However, the consideration of energy prices can lead to the fact that strong branching and pseudo-cost branching become efficient. However, the initialization of strong and pseudo-cost branching is time-consuming. However, variable branching can be efficient if the task variables are almost integral and only a few processing starts need to be fixed.

Analyzing the Workload

Branch-and-bound algorithms are used to divide the problem into smaller subproblems. Within this divide and conquer approach, it is preferable if fewer subproblems must be solved. The number of solved subproblems influences the total solution time. Of course, this idea works well if we can compute near-optimal primal bounds early on. Another often-used strategy is to generate branchings, where one branch directly leads to an infeasible subproblem. Using this branching in the case of job-shop scheduling with flexible energy prices and time windows will be inefficient since we already try to detect infeasibilities in the propagation step of our algorithm. Thus, we are not able to detect further infeasibilities within our branching algorithm.

The example of a fractional solution is shown in Figure 4.11.

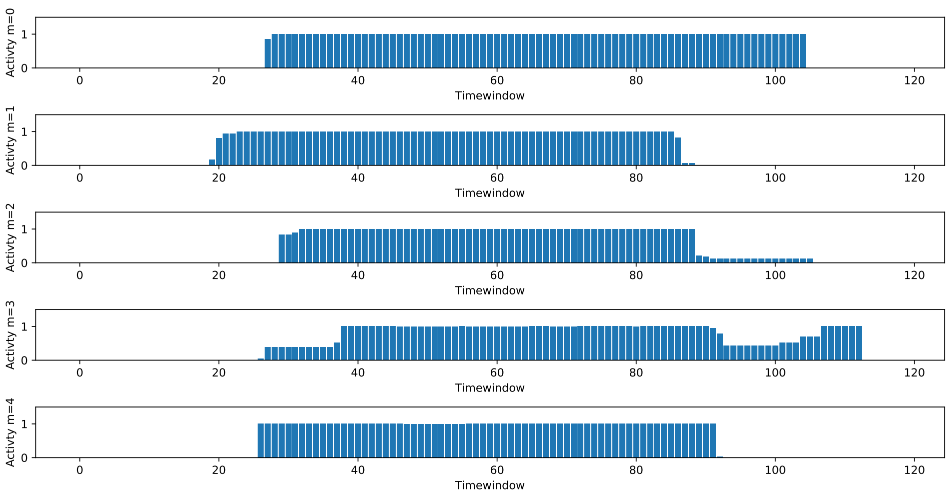


Figure 4.11: This figure shows the workload of the root relaxation of instance $la01\_7\_s\_s$. The workload is partially fractional. The information on the workload is insufficient to generate a workload of an integer feasible solution algorithmically.

The purpose of the branch-and-bound algorithm is not just to fix the fractional aspect of integral parts of LP relaxations. The algorithm is also designed to improve the dual bound until it meets the primal bound. Even in a fractional solution, the setup and processing of each task $(j, k) \in O$ must be completed. Therefore, for each task (j, k), the energy price for $d^{se}_{j,k} + d^{pr}_{j,k}$ many periods is paid, even if this number of periods is composed of many fractional periods. Furthermore, the machine undergoes a complete ramp-up and a complete ramp-down once. Hence, the solution must pay the energy costs for the initial and the final ramping. However, the objective value of the fractional solution does not equal the objective of the optimal integer feasible solution.

The fractional workload and the spread processing starts of the tasks cause the gap between the dual bound and the objective of the optimal solution.

- The fractional workload enables the machine to gradually increase its activity as needed and decrease it during expensive periods. This cost-effective approach generates savings in the more costly periods. An integral solution needs to be either active or inactive in those periods. Therefore, the machine has to pay for the complete ramping, which may become more expensive than the savings within the expensive periods. This is why fractional workload is a crucial challenge to address.
- If processing starts of the tasks are distributed and spread, the tasks can start processing at more preferable times based on the precedence order of their job sequence. This allows for more efficient scheduling of each task. In combination with the fractional workload, the tasks can start in each period, and this behavior is not penalized by the objective since the machine need not ramp up completely to allow the processing to start. However, editing the merged processing does not affect the dual bound. One can imagine that the workload is still fractional, and a reordering of the tasks can lead to a similar workload, and the branching decision, therefore, had no effect.

If the energy prices are positive and ramping is more energy-consuming than *processing* or *setup*, then the machine avoids running completely in active machine states. Since the tasks must be processed, these setup and processing periods are widely distributed, such that the total energy price for processing and setup, combined with the corresponding ramping, is as cheap as possible. The machine state *standby* is used like slack. If the machine is not processing or setting up, and the ramp-down is too expensive, then the fractional usage of standby can be the best choice. The fractional solution tries only to process and set up the tasks (fractionally) and avoids the full ramp-up of the machine. Standby is avoided as much as possible if the corresponding price is positive. In addition, the standby variables get integral values if the task variables are integral, see Theorem 2.6.9.

The priority of our branching is to arrange the active and inactive machine states in each period. Thus, primarily a branching on the workload of a machine $m \in M$ and a period $t \in [T[_{\mathbb{Z}}$ with $0 < w_{m,t} < 1$ prevents the machines from running fractionally in active and inactive machine states simultaneously. Thus, the standby usage is partially increased since fractional usage of ramping is forbidden. Also, processing starts and ramping in preferred (total energy price efficient) periods cannot be done simultaneously. Therefore, the dual bound is strengthened by enforcing the integrality of the workload in both child nodes. In the second step, if the machine's workload is (nearly) integral, the integrality of the active machine states is attached. Therefore, the time windows of the tasks are shrunk until the variables have integral values or no solution can be computed anymore.

Before introducing the branching rules, we state the following facts about feasible integer solutions of the job-shop scheduling problem with flexible energy prices and time windows.

Lemma 4.2.7. *For each integral feasible solution of $\mathcal{P}^B$, the workload $w_{m,t}$ is in $\{0,1\}$ for each $m \in M$ and $t \in [T[_{\mathbb{Z}}$.*

This is derived directly by the formula of the workload and the property that a sum of integral values is integral. Another fact that each feasible integer solution of $\mathcal{P}^B$ satisfies is the following one.

Lemma 4.2.8. *Let $(x, z^{rd,ru}, z^{st}) \in \mathcal{P}^B$ be a feasible integer solution. Then,*

$$\left|\left\{t \in [a_{j,k}, f_{j,k}[_{\mathbb{Z}} \;\mid\; x_{j,k,t} > 0\right\}\right| = 1$$

holds for each $(j,k) \in O$.

Both characteristics are commonly not present in fractional solutions. Therefore, both characteristics are enforced by our branching, which will be introduced in the next section, to generate feasible integer solutions while improving the dual bound.

4.2.3 Workload Branching

The prioritized branching scheme aims at the integrality of the workload. Thus, this branching rule prevents the machines from keeping a fractional workload. The underlying idea is to first determine at what times the machine is *on* and at what times the machine is in a break. Only in the second step does the machine process which task within the online periods.

The idea of this branching rule is to choose one machine $m \in M$ and one period $t \in [T[_{\mathbb{Z}}$ satisfying $0 < w_{m,t} < 1$. Then, the branching rule creates two branch-and-bound nodes A and B with the following consequences.

1. At node A, the machine m is forced to be active in period t. Thus, the usage of breaks in period t is forbidden, and the fractional solution has to change so that the machine is 100% active in period t. Thus, at least the standby usage must be increased in period t. Therefore, the objective value could increase since the induced usage of standby increases the objective value. But there also can exist the case that the fractional solution can be shifted in time, such that the branching condition is satisfied.
2. At child node B, the machine m is forced to be inactive in period t. Since the workload $w_{m,t}$ is fractional in period t and the machine's activity is forced to zero, the local fractional processing and setup have to disappear. If a task is being processed or set up in period t, then the task must be rescheduled, which may result in a more expensive outcome if the task must be processed in a less desirable period. Moreover, some rescheduling on the complete machine could be necessary.

This simplified representation of workload branching illustrates why this branching is considered to drive the dual bound. There are examples of objectives for which this idea of branching will fail, and the improvement of the dual bound is low. In addition, large time windows and nearly constant energy prices could also be difficult to handle. By cleverly shifting the fractional solution within the time window, one can avoid the effect of the branching decisions. The localization of the workload in the time window on each machine is only clearly defined after multiple branch calculations. The consequence can as well as all other known branching rules run into troubles for specific instances.

The workload branching is implemented by imposing the following inequalities, one to each child node:

$$\sum_{(j,k)\in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t+d^{se}_{j,k},} x_{j,k,q} + z^{st}_{m,t} = 1 \text{ at node A} \tag{4.39}$$

$$\sum_{(j,k)\in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t+d^{se}_{j,k}} x_{j,k,q} + z^{st}_{m,t} = 0 \text{ at node B} \tag{4.40}$$

for a well-chosen $m \in M$ and $t \in [T[_{\mathbb{Z}}$.
Constraint (4.39) describes that the machine m cannot use any break in period t locally. Constraint (4.40) describes that the machine must use breaks in period t. The inclusion of the constraint (4.39) to the child node A and (4.40) to the child node B forces the machine activity to be integral in period t on machine m. Nevertheless, choosing $t \in [T[_{\mathbb{Z}}$ and $m \in M$ wisely is important to drive the dual bound.

To choose the machine $m \in M$ and the period $t \in [T[_{\mathbb{Z}}$, such that the impact of the branching is high, the fractionality of the workload needs to be reduced as much as possible in both child nodes.

Definition 4.2.9. *Let $m \in M$ a machine and $w_{m,t}$ the workload of feasible solution $(x, z^{rd,ru}, z^{st}) \in \mathcal{P}^R_{LP}$. Then, the set of all intervals of consecutive fractional activity on machine $m \in M$ is defined as*

$$Q_m := \left\{(q_0, q_1) \in [T[_{\mathbb{Z}} \times [T[_{\mathbb{Z}} \;\middle|\; w_{m,t} - \lfloor w_{m,t} \rfloor > 0 \;\forall\; t \in [q_0, q_1 + 1[_{\mathbb{Z}} \right\}.$$

Intervals of consecutive fractional activity on a machine are interesting because fixing the activity to one within the interval forbids multiple breaks, which are used for many periods. Moreover, the simultaneous fixation of these breaks will, in the best case, lead to an integral workload within the whole interval. On the other hand, the enforcement of using a break in period t within the interval also could lead to the case that only the best break is fixed to one, and the complete fractional processing is shifted into other parts of the fractional schedule. The effect will be more significant if the interval is larger.

A good choice of $m \in M$ and $(q_0, q_1) \in Q_m$ is crucial if multiple intervals of consecutive fractional machine activity exist. We devise the following rules to compute a promising interval of consecutive fractional workload:

- Longest-interval:

$$(\hat{m}, q_0, q_1) = \operatorname*{argmax}_{m \in M,\ (t_0,t_1) \in Q_m} t_1 - t_0.$$

  This rule aims for the longest interval of fractional workload. This rule requires the workload to be always fractional within the intervals. However, the amount of fractionality is not considered, which could lead to bad branches because long fractional intervals are particularly preferred to more fractional intervals.
- Most fractional interval:

$$(\hat{m}, q_0, q_1) = \operatorname*{argmax}_{m \in M,\ (t_0,t_1) \in Q_m} \sum_{t=t_0}^{t_1} \min\{w_{m,t}, 1 - w_{m,t}\}.$$

  In contrast to the longest interval rule, this rule searches for the interval with the largest sum of fractionality. The interval length is not considered directly.
- Uncertainty of the workloads integrality:

$$(\hat{m}, q_0, q_1) = \operatorname*{argmax}_{m \in M,\ (t_0,t_1) \in Q_m} (t_1 - t_0) \cdot \frac{(t_1 - t_0)}{\sum_{t=t_0}^{t_1} \min\{w_{m,t}, 1 - w_{m,t}\}}.$$

  This rule is motivated by the interpretation that a machine must always be active within an interval if the workload is 1 for each period of the considered interval. Also, if the machine is always inactive within an interval, then we are sure that the machine must be inactive within the interval.
  In each period $t \in [t_0, t_1 + 1[_{\mathbb{Z}}$, the coefficient $\frac{(t_1-t_0)}{\sum_{t=t_0}^{t_1} \min\{w_{m,t},1-w_{m,t}\}}$ describes whether the interval $[t_0, t_1]$ is highly fractional or not. Intervals with more fractionality are more favorable than intervals with less fractionality.

Even more complex strategies are possible. However, the computational effort is too high and thus, this branching cannot be used to solve realistic instances. One implemented example is a variant of constraint-based strong branching, which is realized by evaluating all branches of the possible candidates and choosing the best candidate. Although this approach has been implemented, it will not receive any further attention in the remainder of this thesis.

The next step is the fixation of the machine's workload to either be active or inactive in a period $t \in [q_0, q_1 + 1[_{\mathbb{Z}}$. Before presenting the derived rules for computing a promising period $t \in [q_0, q_1 + 1[_{\mathbb{Z}}$, an example motivates the selection rule.

Example 4.2.10. *This illustration is only intended as an example, and there are instances where the solution behaves differently. The sole purpose of this example is to explain the decision-making process for the implemented branching.*

In this example, we consider a fractional solution, analyze possible branching periods within intervals of fractional processing and discuss the resulting changes in the solution. We assume in this example that the effect of the branching can be seen within the interval $[t_0, t_1[_{\mathbb{Z}}$ *such that the results can be visualized. However, this need not be true for all instances, and the optimal fractional solutions resulting in the child nodes can be completely different.*

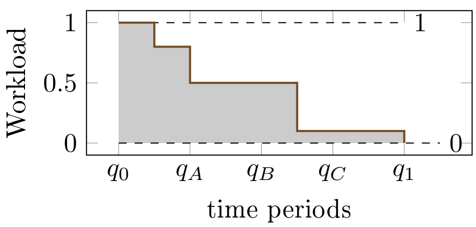


Figure 4.12: Example of a fractional workload.

This example will explore three potential branching periods, denoted as $t' \in \{q_A, q_B, q_C\}$.

1. *The fixation of the workload in period* $t' = q_A$ *is a branch well suited for generating feasible solutions and depth-first search algorithms. The fixation of the workload to be active confirms the proposed solution and only changes the workload such that the solution hopefully becomes more integral. However, the fixation of the workload to zero in period* q_A *is associated with a lot of computational effort. The machine needs to reallocate the processing and setup of the affected tasks and the standby in period*

q_A into neighboring periods to recreate a feasible fractional solution. The fixation to zero cannot prevent the workload from becoming fractional in period q_B.

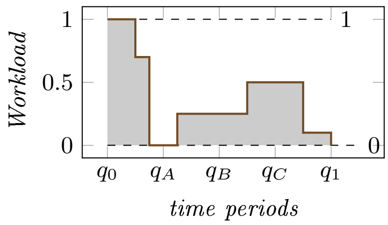


Figure 4.13: Approximation of fractional solution after fixing $w_{m,q_A} = 0$.

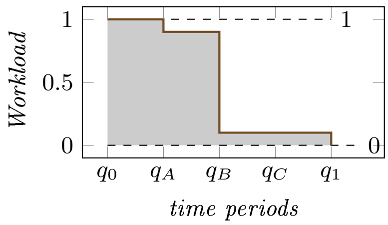


Figure 4.14: Approximation of fractional solution after fixing $w_{m,q_A} = 1$.

2. *The fixation in period t' = q_C creates a branch, which is suitable for generating fractional solutions that do not lead to near-optimal integer solutions since the recreation of a feasible fractional solution requires a large shifting of the fractional solution within the time window or an amount of standby usage to fill the gap between the integral activity of the machine and the fractional usage. Moreover, the fixation to zero has nearly no effect. The fractional processing and setup of tasks or standby will be shifted to the left or, even worse, to the right. In addition, the fractional workload need not change for the most part.*

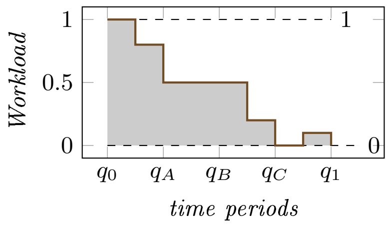


Figure 4.15: Approximation of fractional solution after fixing $w_{m,q_C} = 0$.

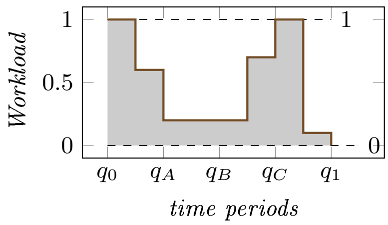


Figure 4.16: Approximation of fractional solution after fixing $w_{m,q_C} = 1$.

3. *The fixation at period t' = q_B is a compromise of a branching in period q_A and a branching in period q_C to balance the workload. This approach ensures that the standby time between t_0 and q_B is not overly costly. Deciding for fixation at zero is also beneficial because it can lead to near-optimal solutions, just like the one-fixation. In short intervals, the fixation will be as effective as visualized. However, in more realistic cases, the branch will result in branches like for t' = q_C since the fractional solution can shift parts.*

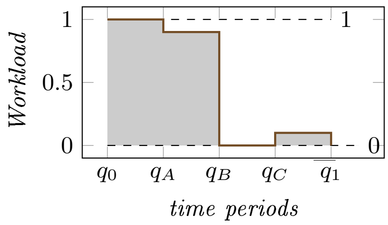


Figure 4.17: Approximation of fractional solution after fixing $w_{m,q_B} = 0$.

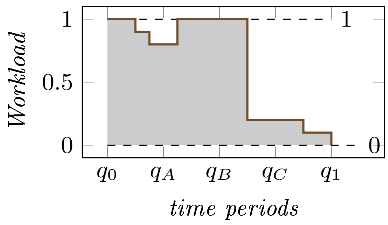


Figure 4.18: Approximation of fractional solution after fixing $w_{m,q_B} = 1$.

However, there are bad examples where the branching will fail. One example is visualized in Figure 4.19.

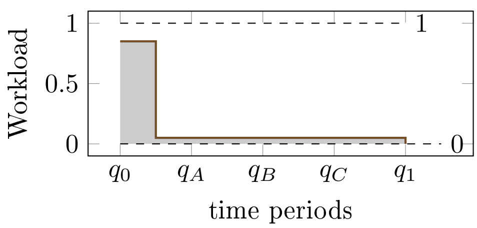


Figure 4.19: Example of a fractional workload, where the branching will fail.

Each of our branchings will not be as strong as wished, except the branch in period $t' = q_A$. The branches $t' \in \{q_B, q_C\}$ will not be good since both branches create, which can lead to similar situations as visualized in Figure 4.19 in the case of the 0*-branch. The case of the* 1 *branch will always improve the dual bound, while the* 0*-branch has nearly no effect. Thus, the fractional solution needs to be analyzed before deciding about a suitable period to branch on.*

Motivated by the examples of the fractional workload, the following rule is proposed to achieve a branching in period q_B:

$$t' = \left\lfloor \frac{\sum_{t=q_0}^{q_1} t \cdot \ w_{m,t}}{\sum_{t=q_0}^{q_1} w_{m,t}} \right\rfloor . \tag{4.41}$$

Note that the valuation of t' is similar to the proposed SOS1-branching scheme of Beale and Tomlin [BT69]. However, we aggregate certain parts of constraints and compute the mean value, while Beale and Tomlin only consider variables of the same constraint.

This rule computes a $t' \in [t_0, t_1[_{\mathbb{Z}} \in Q_m$ with $0 < w_{m,t'} < 1$. Moreover, this weighted mean considers the position of the main focus of the fractional workload as well as the length of the fractional workload. Thus, the generation of feasible solutions and the disallowing of directions of expensive solutions is supported. The branching period will be near the period q_B, and the branching will mostly be effective.

Some estimators for the improvement of the dual bound after specific branchings were considered. However, the estimators currently in use either do not provide much useful information or need a lot of computational power. That's why we prefer using combinatorial conditions to decide on branching.

The idea of an estimator of the change of the fractional solution can be computed as follows.

- If the machine m is fixed to be active in period t', then all breaks $(t_0, t_1) \in B_m$ with $z^{rd,ru}_{m,t_0,t_1} > 0$ and $t' \in [t_0, t_1[_{\mathbb{Z}}$ are fixed to zero. Then, after resolving the child node, at least the change of the fractional solution after the fixation to one results in $\sum_{(t_0,t_1)\in B_m} (t_1 - t_0) z^{rd,ru}_{m,t_0,t_1}$.
- If the machine m is fixed to be inactive in period t', then all tasks $(j, k) \in O^M_{|m}$ cannot start processing or setup, such that the task is affecting period t'. Moreover, there is no standby usage in period t' allowed.

Thus, the change of the solution in both child nodes can be estimated by

$$\begin{aligned} I(m,t) = \min\{ &\sum_{(j,k)\in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}-1}^{t+d^{se}_{j,k}} (d^{se}_{j,k} + d^{pr}_{j,k}) x_{j,k,q} + z^{st}_{m,t}), \\ &\sum_{(q_0,q_1)\in B_m : t\in[q_0,q_1[_{\mathbb{Z}}} (q_1 - q_0) z^{rd,ru}_{m,q_0,q_1} \}. \end{aligned}$$

Theorem 4.2.11. *Let $m \in M$ be one machine and $t \in [T[_{\mathbb{Z}}$ be a period. Then $I(m, t) = 0$ holds, if the workload $w_{m,t}$ is integral.*

This selection rule did not prevail in our first experiments, and therefore, this selection rule is only mentioned to document the thought processes in this research direction.

Thus, one can choose $t' = \text{argmax}\{I(m, t) \mid t \in [t_0, t_1, \}[_{\mathbb{Z}}$. Nevertheless, we want to consider the length of the interval. Therefore, the third way of computing the branching period can be done by

$$t' = \left\lfloor \frac{\sum_{t=t_0}^{t_1} t \cdot \ I(m,t)}{\sum_{t=q_0}^{q_1} I(m,t)} \right\rfloor . \tag{4.42}$$

The proposed branching is valid and creates a real disjunction.

Theorem 4.2.12. *The branching* (4.40) *and* (4.39) *with the proposed selection rules is an SOS branching.*

Proof. Let $\hat{m} \in M$ and $t' \in [T[_{\mathbb{Z}}$, such that $w_{\hat{m},t'} - \lfloor w_{\hat{m},t'} \rfloor > 0$. Thus, the workload on machine $\hat{m}$ in period t is fractional, while the constraint

$$\sum_{(j,k)\in O^M_{|\hat{m}}} \sum_{q=t'-d^{pr}_{j,k}+1}^{t'+d^{se}_{j,k}} x_{j,k,q} + z^{st}_{\hat{m},t'} + \sum_{\substack{(t_0,t_1)\in B_{\hat{m}}: \\ t'\in\{t_0,\ldots,t_1\}}} z^{rd,ru}_{m,t_0,t_1} = 1$$

is satisfied. Dividing the set S of the, in the equality, present variables, such that the set $S_1 = \{z^{rd,ru}_{m,t_0,t_1} : (t_0,t_1) \in B_{\hat{m}}$ and $t' \in [t_0, t_1+1[_{\mathbb{Z}}\ \}$ and $S_2 = \{x_{j,k,t} \mid (j,k) \in O^M_{|m}$ and $t \in [t' - d^{pr}_{j,k} + 1, t' + d^{se}_{j,k}[_{\mathbb{Z}}\ \} \cup \{z^{st}_{\hat{m},t'}\}$. If we sum up all contained variables, the quantities S_1 and S_2 have a real positive value. Therefore, the decision to use S_1 or S_2 to cover $\hat{m}$ in period t' will be a branching. □

By usage of this branching rule, the workload profile gets defined. The fixation of the processing starts of the tasks within the specified workload is part of a second branching rule. However, the information about the processing starts is not mandatory to prune. Moreover, this branching increases the local dual bound efficiently at least in one child node and in combination with a near-optimal solution, the branch-and-bound tree will still be small.

The branching is only useful if some conditions are satisfied.

Observation 1. *Let $m \in M$ and $[q_0, q_1[_{\mathbb{Z}} \in Q_m$ and $t \in [q_0, q_1[_{\mathbb{Z}}$. The branching on the machine's activity in period t of machine m is successful, if the workload in the interval $[q_0, q_1[_{\mathbb{Z}}$ satisfies:*

1. $q_1 - q_0 > 0$
2. $\epsilon < w_{m,t} < 1 - \epsilon$

with $\epsilon \in (0, 0.5]$.

In our implementation, we propose to use $\epsilon = 0.01$ to allow branchings, which fix the assumed workload direction.

In addition, the branching can be ineffective, although the workload is not integral yet. It is not necessary to perform workload branching until the workload of machine m is integral in each period $t \in [T[_{\mathbb{Z}}$.

4.2.4 Branching on Assignment Constraints

Considering scheduling in integer programming, a well-known branching rule for the time-indexed formulation is to branch on assignment constraints (3.10b), a variation of SOS branching. One example of SOS branching for time-indexed variables is explained in [vdA94]. This rule shrinks the time windows of the tasks with fractional processing starts such that the processing starts either do not exist (infeasible node) or the processing starts are integral. Therefore, we need to find tasks with fractional start variables.

Definition 4.2.13. *Let $(x, z^{st}, z^{rd,ru}) \in \mathcal{P}^B_{LP}$ be a feasible solution and $(j,k) \in O$ a task. The parameters*

$$l(j,k) = \min\{t \in [T[_{\mathbb{Z}} \mid\ x_{j,k,t} > 0\}$$

and

$$r(j,k) = \max\{t \in [T[_{\mathbb{Z}} \mid\ x_{j,k,t} > 0\}$$

describe the first and the last fractional processing start of task (j,k).

Definition 4.2.14. *A task $(j,k) \in O$ with $r(j,k) - l(j,k) > 1$ is called **a fractional task**. The task satisfies $|\{t \in [T[_{\mathbb{Z}} \mid\ x_{j,k,t} > 0\}| \geq 2$.*

The assignment constraint branching rule searches for a fractional task $(\hat{j}, \hat{k}) \in O$. The fractionality of the task indicates the existence of at least two distinct periods $t_1, t_2 \in [T[_{\mathbb{Z}}$ with $x_{\hat{j},\hat{k},t_1} > 0$ and $x_{\hat{j},\hat{k},t_2} > 0$. Then, the set of allowed processing starts $V = [a_{\hat{j},\hat{k}}, f_{\hat{j},\hat{k}}[_{\mathbb{Z}}$ of task $(\hat{j}, \hat{k})$ is divided into the disjunctive subsets $V_1 := [a_{\hat{j},\hat{k}}, t' + 1[_{\mathbb{Z}}$ and $V_2 := [t' + 1, f_{\hat{j},\hat{k}}[_{\mathbb{Z}}$ with a suitable choice $t' \in [l(\hat{j},\hat{k}), r(\hat{j},\hat{k})[_{\mathbb{Z}}$. The branching is devised by creating two child nodes A and B with the following branching constraints

$$\sum_{t=0}^{t'} x_{j,k,t} = 0 \text{ at node } A \text{ and } \sum_{t=t'+1}^{T} x_{j,k,t} = 0 \text{ at node } B. \tag{4.43}$$

The choice of the fractional task and the branching period t' highly influence the performance of the branch-and-bound algorithm. Also, slight differences within the choice of the period t' can lead to completely different behavior of the solution process. Additionally, the enhancement of the dual bound and the efficiency of resolving the relaxation are closely linked to the choice of the fractional task.

Choosing the most promising task is crucial for successfully applying the branching rule. The set of all fractional tasks is defined by

$$O^{frac} := \{(j,k) \in O \mid r(j,k) - l(j,k) \geq 1\}.$$

We consider the following aspects of fractional tasks while searching for them:

- The spread of the fractional processing. An integer feasible solutions satisfy

$$r(j,k) + 1 - l(j,k) = 1 \text{ for each } (j,k) \in O.$$

  If $r(j,k) + 1 - l(j,k) > 1$ holds, then the processing of the task (j,k) is not fixed. Thus, tasks $(j,k) \in O$, where $r(j,k) + 1 - l(j,k)$ is large are more favorable than tasks, where $r(j,k)+1-l(j,k)$ is small since latter is nearly fixed to a certain period.
- The number of interruptions: if the fractional processing of task (j,k) happens simultaneously with many fractional usage of breaks and standby and further fractional tasks, it seems to be more important to branch on this task to organize and tidy up the fractional solution. Only afterward, the fractional tasks, which are fractionally processed in a relatively isolated way, are considered to be branching candidates. These solutions are also candidates for variable branching

While computing a promising branching candidate, the aspects of a promising fractional task are considered. The period t' is computed afterward by a second rule.

- GUB-Dichotomy: To choose the task with the largest spread, select

$$(\hat{j},\hat{k}) = \operatorname*{argmax}_{(j,k)\in O} r(j,k) - l(j,k).$$

  This rule has been previously mentioned in [vdA94] and selects the fractional tasks whose fractional variables are distributed most widely.
- Maximum propagation: choose the task $(\hat{j},\hat{k}) \in O$ and possibly also the period $t' \in [l(j,k), r(j,k)[_{\mathbb{Z}}$ with the largest number of variable reductions in both child nodes by propagation and cutoffs. This rule is computationally expensive since the propagation algorithm needs to be used for each branching candidate. It is often the case that the variable reductions only affect the concerned task and the surrounding tasks in the associated job sequence. In this case, the approach resembles "choose the task with the longest processing and set-up time," which is not our intention.
- Widely interrupted tasks (WI): choose the task

$$(\hat{j},\hat{k}) := \operatorname*{argmax}_{(j,k)\in O} \left\{ \Big(r(j,k) - l(j,k)\Big) \cdot \sum_{(i,l\in O^M_{|m_{j,k}}} \sum_{t=l(j,k)}^{r(j,k)} x_{i,l,t} \right\}.$$

  This branching rule combines the aspects of a meaningful branching rule in the case of assignment constraint branching. This branching rule searches for tasks, violating properties, which are identified to be important for feasible integer solutions. The branching will repair those properties, and hopefully, the search for near-optimal solutions will be supported.
- Long and isolated tasks (LI): choose the task

$$(\hat{j},\hat{k}) := \operatorname*{argmax}_{(j,k)\in O} \left\{ \frac{\Big(r(j,k) - l(j,k)\Big)^2}{\sum_{(i,l)\in O^M_{|m_{j,k}}} \sum_{t=l(j,k)}^{r(j,k)} x_{i,l,t}} \right\}.$$

  This branching rule selects tasks whose fractional processing is not interrupted very often by further tasks. Thus, resolving the child nodes may be fast since, hopefully, only one task needs to be rescheduled.

The rule of long and isolated tasks is applicable in the case where we expect to detect many integral solutions since the solution is changed mainly for tasks with a nearly fixed processing start. The rule of wide and interrupted tasks should be used in the beginning to tidy up the fractional solution. However, this requires an internal analysis of the fractional solutions with well-trained parameters and indicators to optimally control the solution process.

Theorem 4.2.15. *If $O^{frac} \neq \emptyset$ holds, then the task selection detects one fractional task, and there exists at least one $t' \in [l(j,k), r(j,k)[_{\mathbb{Z}}$ such that we can create a valid branching.*

Proof. Let $|O^{frac}| > 0$ and $(j,k) \in O^{frac}$. There exists one $t' \in [l(j,k), r(j,k)[_{\mathbb{Z}}$ with

$$\sum_{t=l(j,k)}^{t'} x_{j,k,t} > 0 \text{ and } \sum_{t=t'}^{r(j,k)} x_{j,k,t} > 0.$$

Thus, the task selection is well-defined and always returns a fractional task if one exists. □

Theorem 3.2.7 indicates that at least one fractional task exists if the current LP-relaxation of the branch-and-bound node is fractional. Consequently, the task's time window will be split into two intervals, valid for the local child node. To complete the branching rule, there is a need for a period $t' \in [l(\hat{\jmath},\hat{k}), r(\hat{\jmath},\hat{k})[_{\mathbb{Z}}$.

Before the presentation of the computation of the period selection, multiple approaches targeting different effects on the fractional solution need to be discussed in the following example.

Example 4.2.16. *We are using the following fractional solution to visualize the different results of the assignment constraint branching.*

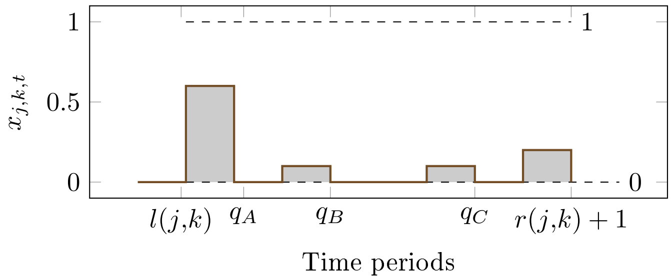


Figure 4.20: Fractional solution of task (j,k).

The task $(\hat{\jmath},\hat{k})$ is fractionally processed in $[l(\hat{\jmath},\hat{k}), r(\hat{\jmath},\hat{k})+1[_{\mathbb{Z}}$.

- *The branch $t' = q_C$ is classified as an unbalanced branch. The dual bound of the resulting child nodes increases only in one child node since this branching would be similar to a variable branching. The branch is classified to be unbalanced since the larger interval $V_1 = [a_{\hat{\jmath},\hat{k}}, q_C[_{\mathbb{Z}}$ also contains the main part of the fractionality. Thus, the fractional solution for V_1 need only shift a small amount of fractionality into V_1. In contrast, the main branch for $V_2 = [q_C+1, f_{\hat{\jmath},\hat{k}}[_{\mathbb{Z}}$ becomes extremely violated, and a large part of the of the fractional processing of task $(\hat{\jmath},\hat{k})$ has to be rescheduled.*
- *In contrast, branching in a period $t' = q_A$ or in period $t' = q_B$ leads to the following disjunctive time windows $V1 = [l(\hat{\jmath},k), t'[_{\mathbb{Z}}$ $V_2 = [t'+1, r(\hat{\jmath},\hat{k})+1[_{\mathbb{Z}}$. The child node using time window V_2 has to reschedule the task $(\hat{\jmath},\hat{k})$ since the main part of the fractionality is formerly located in forbidden periods. The branch in period q_B is estimated to be similar to that in period q_A. Moreover, the branch balances the number of variable reductions and fractionality in both child nodes. Thus, we consider the branching in period q_B to be favorable.*

The situation is easier if the fractional solution is similar to the one visualized in Figure 4.21.

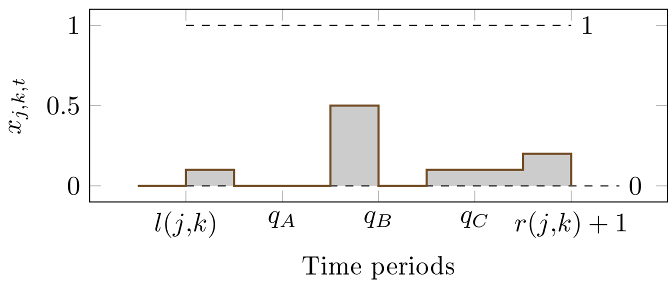


Figure 4.21: Fractional solution of task (j,k).

In that case, the period t' should be set so that period q_B is used. Then, the number of fractionality and domain reductions is balanced in both branches.

The following branching rules consider the mentioned aspects of a balanced assignment constraint branching.

- Weighted mean (WM): choose

$$t' = \sum_{t=l(j,k)}^{r(j,k)} x_{\hat{\jmath},\hat{k},t}\, t$$

to split the interval concerning the distribution of the fractional values of task $(\hat{j}, \hat{k})$. By computing the weighted mean, one can exactly cluster points within the interval $[l(j,k), r(j,k)[_{\mathbb{Z}}$, which may indicate the location of the local optimal integer feasible solution. This technique was introduced in [vdA94].
- Propagation score: choose $t' \in [l(j,k), r(j,k)[_{\mathbb{Z}}$ such that the resulting number of domain reductions by propagation is maximum. As mentioned, the number of domain reductions is often very similar for multiple periods and, therefore, insignificant.
- Constraint violation (CV): choose

$$t' = \max_{t \in [l(\hat{j},\hat{k}), r(\hat{j},\hat{k})[_{\mathbb{Z}}} \min\Big\{(r(\hat{j},\hat{k}) + 1 - t) \cdot \sum_{q=l(\hat{j},\hat{k})}^{t} x_{\hat{j},\hat{k},q}, (t - l(\hat{j},\hat{k})) \cdot \sum_{q=t+1}^{r(\hat{j},\hat{k})} x_{\hat{j},\hat{k},q}\Big\}.$$

The computed period t' is computed by the product of the reduced interval's size, which may not match the task's real-time window and the violation of the branching constraint in the child node. The maximum of the minimum describes a branching period, leading to a balanced branching and maybe balanced progress in both child nodes.

Theorem 4.2.17. *The branching* (4.40) *and* (4.39) *with the proposed selection rules and is an SOS branching and thus a valid branching, if* $(\hat{j}, \hat{k}) \in O$ *satisfies* $|\{x_{\hat{j},\hat{k},t} | t \in [T[_{\mathbb{Z}} \}| > 1$ *and* $t \in [l(\hat{j},\hat{k}), r(\hat{j},\hat{k})[_{\mathbb{Z}}$ *.*

The branching rule by propagation is totally ignored in the following and also in our experimental results. The branching candidate itself mainly influences the corresponding branching score. The branching rule by constraint violation is motivated by an analysis of the resulting subproblems since the violation of the time window of task $(\hat{j}, \hat{k})$ and the distribution of the fractionality are equally considered.

4.2.5 Branching Rule Selection

The workload branching is the prioritized branching rule since it is designed to be dual bound driving by default. The branching on assignments constraint is only the second choice since this branching only schedules the fractional task variables without considering the workload and the corresponding energy costs. However, sometimes, the workload of the machines is still fractional, and the assignment constraint branching should be the preferred branching rule to improve the dual bound. This is the case if the workload does not define the objective value in particular. Moreover, the fractionality of the tasks leads to a minimum gap between the optimal objective value and the fractional solution and reordering the tasks closes the gap.

To only use the workload branching if it is promising, we decided to skip the workload branching and directly use assignment constraint branching in some cases. If the workload of all machines is nearly integral, then the workload branching is ineffective, and assignment constraint branching is more efficient since the change of the fractional solutions becomes small.

The following expression verifies the near integrality condition:

$$\sum_{m \in M} \sum_{t \in [T[_{\mathbb{Z}}} \min\{1 - w_{m,t}, w_{m,t}\} < 0.1 \cdot \sum_{m \in M} \sum_{t \in [T[_{\mathbb{Z}}} w_{m,t}.$$

In addition, the workload branching is inefficient if the workload's fractionality is mainly located on one machine.

We decide not to branch on machine state constraints if the following condition is satisfied:

$$\sum_{m \in M} \sum_{t \in [T[_{\mathbb{Z}}} \min\{w_{m,t}, 1 - w_{m,t}\} \cdot \beta > \max_{m \in M} \sum_{t \in [T[_{\mathbb{Z}}} \min\{w_{m,t}, 1 - w_{m,t}\}.$$

The value β is chosen in $[0.3, 0, 6]$ to describe that about half of the fractionality can be located on the workload profile of one single machine.

4.3 Separation of Valid Inequalities

Within the branch-and-bound-and-cut algorithm, we consecutively solve the LP-relaxation of the branch-and-bound nodes until the considered nodes can be pruned, are infeasible, or the optimal integer feasible solution for the subproblem is detected. To strengthen the description of the feasible solution, further valid but violated constraints are added to the problem formulation to tighten the LP-relaxation. Well-known techniques to strengthen the LP-relaxation at the current branch-and-bound are mentioned, for example, in [CCZ14].

We will present different classes of cutting planes within the context of the job-shop scheduling problem with flexible energy prices and time windows. This section starts with the conflict graph and clique cuts. After that, we extend known GUB cover constraints for single-machine scheduling by break variables. Then, we derive valid inequalities from the linear ordering problem.

4.3.1 Conflicts and Clique Cuts

Clique cuts and conflicts are well-known techniques in integer programming. The detection of conflicts and the separation of clique cuts are implemented in many commercial solvers, for example, Gurobi [ABG+20, p.28]. The concept of the conflict graph, the implementation, and the algorithmic approaches are described in [ANS00]. A conflict graph is a mathematical structure describing subsets of binary variables of an ILP that cannot be simultaneously equal to 1 in any feasible solution.

Definition 4.3.1. *We are given the integer linear program*

$$\min\{c^\top x \mid \ Ax \leq b, x \in \{0,1\}^n\},$$

with $c \in \mathbb{Z}^n,\ A \in \mathbb{Z}^{m\times n}, b \in \mathbb{Z}^m$. *A conflict graph* $G = (V,E)$ *is an undirected graph* G, *with* $V = \{x_j \mid j = 1,\dots,n\} \cup \{\overline{x}_j \mid j = 1,\dots,n\}$ *and* $E \subseteq \binom{V}{2}$ *and for each* $\{u,w\} \in E$ *the inequality* $u + w \leq 1$ *is valid for* $Ax \leq b,\ x \in \{0,1\}$.

The edges of the conflict graph visualize the pairwise conflicts of variables and negations of variables. The vertex $x_j = u \in V$ describes the setting of variables x_j to 1. The vertex $\overline{x}_j = v \in V$ describes the setting of x_j to 0. The logical relations and corresponding linear constraints are presented in Table 4.1.

Table 4.1: Possible edges within the conflict graph and their constraint representation.

edge	conflict cut
$\{x_j, x_i\}$	$x_j + x_i \leq 1$
$\{\overline{x}_j, \overline{x}_i\}$	$x_j + x_i \geq 1$
$\{\overline{x}_j, x_i\}$	$(1 - x_j) + x_i \leq 1$
$\{x_j, \overline{x}_i\}$	$x_j + (1 - x_i) \leq 1$

Many conflicts can be directly detected within the model if there are set-partitioning or set-packing constraints [ABG+20, p.28]. More edges can be detected by probing within the presolving stage. Here, many variables are consecutively fixed to zero and one. If the problem becomes infeasible, one tries to describe the reason for the infeasibility using a conflict in the conflict graph. Atamturk et al. describe a method to detect infeasibilities in [ANS00, p.42]. A more complex topic is conflicts describing variable constellations that are not possible in any optimal solutions. It is also a valid approach to describe clique cuts, cutting off non-optimal solutions. However, it is crucial that at least one optimal solution remains feasible.

Definition 4.3.2 (Clique and stable set)**.** *Let* $G = (V,E)$ *be an undirected graph. A set* $C \subseteq V$ *is called* ***clique****, if for each pair* $u, v \in C$ *the edge* $\{u,v\}$ *exists in* E.
A set $S \subseteq V$ *is called* ***stable set****, if for each pair* $u, v \in S$ *the edge* $\{u,v\}$ *does not exists in* E.

For a clique $C \subseteq V$, with $C = I \dot{\cup} \overline{I}$, the corresponding **clique cut** is defined by

$$\sum_{i\in I} x_i + \sum_{i\in \overline{I}} (1 - x_i) \leq 1.$$

The problem $Ax \leq b$, $x \in \{0,1\}^n$ may indicate the stable set problem as a subproblem. Different variables cannot be simultaneously equal to 1. Since the clique cut is a valid constraint of the stable set polyhedron, the clique cut is a valid constraint of $Ax \leq b$ with $x \in \{0,1\}$.

The phrase "Add a conflict to the conflict graph" describes an extension of the conflict graph. In the case of the conflict $\sum_{i\in I} x_i + \sum_{i\in \overline{I}}(1-x_i) \leq 1$, the conflict graph is extended as follows:

$$\begin{aligned} E \Leftarrow E &\cup \Big\{\{x_i, x_j\}\Big| i,j \in I,\ i \neq j\Big\} \\ &\cup \Big\{\{\overline{x}_i, \overline{x}_j\}\Big| i,j \in \overline{I}, i \neq j\Big\} \\ &\cup \Big\{\{x_i, \overline{x}_j\}\Big| i \in I,\ j \in \overline{I}\Big\}. \end{aligned}$$

The linear constraint $\sum_{i\in I} x_i + \sum_{i\in \overline{I}}(1-x_i) \leq 1$ is referred to as a conflict as different variables cannot be equal to one at the same time.

The size of the conflict graph grows with the number of added conflicts. Moreover, the size of the conflict graph highly influences efficiency of the detection of violated conflict cuts. The detection of the most violated clique cut, represented within the conflict graph, is *NP*-hard. This clique is detected by a maximum-weighted clique algorithm, for example, the TClique-algorithm [BK97] by Borndörfer and Kormos. The TClique-algorithm is an exact algorithm, which also can be used heuristically by not enumerating the whole branch-and-bound tree [Ach09, p.110]. Thus, the search for violated cliques can be done by an efficient heuristic.

The conflict graph of the job-shop scheduling problem with flexible energy prices and time windows initially contains the constraints (3.10b), (3.10c) and (3.10d). More conflicts can be detected by probing, which could be time-consuming because there are $\mathcal{O}(T^2 \cdot n_M)$ many binary variables. Therefore, the knowledge of valid conflicts speeds up the conflict graph generation. The following section contains conditions for valid conflicts of the job-shop scheduling problem with flexible energy prices and time windows and the proof of their validity.

Conflicts of Task Variables

Within this section, we mention valid conflicts only considering the task variables.

Theorem 4.3.3 (Conflict by a fixed processing start). *Let $(j,k),(i,l) \in O^M_{|m}$ be two distinct tasks processed by machine $m \in M$. For each period $t \in [T[_{\mathbb{Z}}$, the constraint*

$$x_{j,k,t} + \sum_{q=t-d^{se}_{j,k}-d^{pr}_{i,l}+1}^{t+d^{pr}_{j,k}+d^{se}_{i,l}-1} x_{i,l,q} \leq 1 \tag{4.44}$$

is a valid constraint of $\mathcal{P}^B$.

Proof. To show that the constraint (4.44) is a valid constraint of the polyhedron $\mathcal{P}^B$, we will show that the constraint is feasible for $x_{j,k,t} = 1$ and also for $x_{j,k,t} = 0$.

If $x_{j,k,t}$ is 1, then the machine is blocked from period $t - d^{se}_{j,k}$ up to $t + d^{pr}_{j,k} - 1$ by processing and setting up task (j,k). The machine $m = m_{j,k}$ cannot simultaneously process or set up two tasks. Therefore, the task $(i,l) \in O^M_{|m} \setminus \{(j,k)\}$ needs to start processing either before task (j,k) or after task (j,k).

1. If the task (i,l) starts its processing before task (j,k), then the processing of (i,l) and the setup of (j,k) need to be completed before period t. Thus, the task (i,l) cannot start within the interval $[t - d^{se}_{j,k} - d^{pr}_{i,l}, t+1[_{\mathbb{Z}}$.
2. If the task (i,l) starts processing after (j,k), then the processing of (j,k) and the setup of (i,l) need to be completed after period $t + d^{pr}_{j,k} - 1$. Thus, the task (i,l) cannot start within the interval $[t, t + d^{pr}_{j,k} - 1 + d^{se}_{i,l} + 1[_{\mathbb{Z}}$.

Therefore, the task $(i),l)$ cannot start processing in any period $q \in [t - d^{se}_{j,k} - d^{pr}_{i,l}, t + d^{pr}_{j,k} - 1 + d^{se}_{i,l} + 1[_{\mathbb{Z}}$, if task (j,k) starts processing in period t and the inequality is valid for $x_{j,k,t} = 1$. I If $x_{j,k,t} = 0$, then the equation

$$x_{j,k,t} + \sum_{q=t-d^{se}_{j,k}-d^{pr}_{i,l}+1}^{t+d^{pr}_{j,k}+d^{se}_{i,l}-1} x_{i,l,q} = 0 + \sum_{q=t-d^{se}_{j,k}-d^{pr}_{i,l}+1}^{t+d^{pr}_{j,k}+d^{se}_{i,l}-1} x_{i,l,q} \leq 1$$

is valid since it is already described by constraint (3.10b). Thus, the constraint (4.44) is valid for all integer feasible solutions of $\mathcal{P}^B$. □

Knapsack Covers of Size Two

Another set of conflicts are covers of the knapsack constraint (3.18), (3.19), (3.20), (3.21), (3.22) of size 2. We can detect the covers by iterating over two lists of breaks and comparing the size of the breaks with the sizes of the knapsacks. To simplify the notation of the knapsack constraints and the corresponding covers, we introduce parameters describing the earliest release date minus the corresponding setup time and the latest due date of a task by

$$a_m^M = \min_{(j,k)\in O^M_{|m}} (a_{j,k} - d^{se}_{j,k})$$

and

$$f_m^M = \max_{(j,k)\in O^M_{|m}} (f_{j,k} - 1 + d^{pr}_{j,k}).$$

The first theorem considers the conflicts derived from the complete time window.

Theorem 4.3.4 (Covers with right-hand-side 1). *Let $m \in M$ be one machine. We are given the knapsack constraint*

$$\sum_{(t_0,t_1)\in B_m} (t_1 - t_0) \cdot z^{rd,ru}_{m,t_0,t_1} \leq T + d^{ru}_m + d^{rd}_m - \sum_{(j,k)\in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k}).$$

Then for each distinct pair of breaks (t_0,t_1), $(t_2,t_3) \in B_m$ satisfying

$$t_1 - t_0 + t_3 - t_2 > T + d^{ru}_m + d^{rd}_m - \sum_{(j,k)\in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k})$$

the linear constraint

$$z^{rd,ru}_{m,t_0,t_1} + z^{rd,ru}_{m,t_2,t_3} \leq 1 \tag{4.45}$$

is a valid constraint of $\mathcal{P}^B$.

The validity of the cover conflicts is derived from the validity of the knapsack-constraint (3.18).

The derived cover describes a combination of breaks that cannot simultaneously be part of a feasible integral solution since using multiple breaks from the cover will prevent at least one task from completing its processing or setup.

A similar statement can be derived for the knapsack constraints of inner breaks (3.21).

Theorem 4.3.5 (Covers of inner breaks). *Let $m \in M$ be one machine. The breaks (t_0,t_1) and $(t_2,t_3) \in B_m$ cannot be simultaneously utilized in any feasible integer solution when the following condition is satisfied:*

$$t_3 - t_2 + t_1 - t_0 > f^M_m - \Big(a^M_m + \sum_{(j,k)\in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k})\Big).$$

Then, the conflict $z^{rd,ru}_{m,t_0,t_1} + z^{rd,ru}_{m,t_1,t_2} \leq 1$ is a valid constraint of $\mathcal{P}^B$.

These conflicts are covers of the knapsack constraints (3.21) and thus valid. The covers of inner breaks can easily be extended to also consider middle, initial and final breaks.

Theorem 4.3.6 (Conflict of initial and middle breaks). *Let $m \in M$ be one machine. Additionally, let $(t_0,t_1),(t_2,t_3) \in B_m$ two distinct breaks. The breaks $(t_0,t_1),(t_2,t_3)$ cannot simultaneously assigned to have value 1 in any feasible solution of $\mathcal{P}^B$ if*

$$\Big|[t_0,t_1[_{\mathbb{Z}} \cap [a^M_m, f^M_m[_{\mathbb{Z}}\Big| + \Big|[t_2,t_3[_{\mathbb{Z}} \cap [a^M_m, f^M_m[_{\mathbb{Z}}\Big| > f^M_m - \Big(a^M_m + \sum_{(j,k)\in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k})\Big)$$

holds. Then, the conflict

$$z^{rd,ru}_{m,t_0,t_1} + z^{rd,ru}_{m,t_2,t_3} \leq 1$$

is a valid constraint of $\mathcal{P}^B$.

Proof. Let $m \in M$ and $(t_0,t_1),(t_2,t_3) \in B_m$ two distinct breaks with $t_0 < t_2$ and $0 < t_2 < t_3 < T$. We consider the knapsack constraints (3.17) for $l = a^M_m$ and $r = f^M_m$. The coefficient of the break (t_0,t_1) within the knapsack constraint is $\pi^B_{t_0,t_1} = \max\Big\{0, \min\{r,t_1\} - \max\{l,t_0\}\Big\} = |[t_0,t_1[_{\mathbb{Z}} \cap [a^M_m, f^M_m[_{\mathbb{Z}}|$. Each task must complete its setup and processing within $[l, r[_{\mathbb{Z}}$. Therefore, we can simplify the right-hand-side to $f^M_m - (a^M_m + \sum_{(j,k)\in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k}))$. Thus, we can consider also the simplified knapsack constraint

$$\sum_{(t_0,t_1)\in B_m} \pi^B_{t_0,t_1} z^{rd,ru}_{m,t_0,t_1} \leq f^M_m - \Big(a^M_m + \sum_{(j,k)\in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k})\Big),$$

which is still valid. The breaks (t_0, t_1) and (t_2, t_3) can be used simultaneously if they do not prevent one task from completing its processing and setup. Thus, the required number of periods by (t_0, t_1) and (t_2, t_3) within $[a^M_m, f^M_m[_{\mathbb{Z}}$ must be smaller than the required space for processing and setting up all the tasks. Suppose the breaks $(t_0, t_1), (t_2, t_3)$ are used simultaneously within a feasible integer solution, and the breaks satisfy

$$\big|[t_0, t_1[_{\mathbb{Z}} \cap [a^M_m, f^M_m[_{\mathbb{Z}}\big| + \big|[t_2, t_3[_{\mathbb{Z}} \cap [a^M_m, f^M_m[_{\mathbb{Z}}\big| > f^M_m - (a^M_m + \sum_{(j,k)\in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k})).$$

1. If $[t_0, t_1[_{\mathbb{Z}} \cap [t_2, t_3[_{\mathbb{Z}} \neq \emptyset$, then both breaks cannot be used simultaneously because the constraints (3.10d), (3.10e) and (3.10f) hold.
2. If $[t_0, t_1[_{\mathbb{Z}} \cap [t_2, t_3[_{\mathbb{Z}} = \emptyset$, then both breaks cannot be used since the combination of the breaks requires too many periods and prevents at least one task in completing its setup and processing.

Thus, the corresponding conflict is valid. □

It is valid to consider subsets of tasks instead of all tasks processed by machine $m \in M$. Then, the following condition holds.

Theorem 4.3.7 (Conflict of initial and middle breaks)**.** *Let $m \in M$ be one machine. Additionally, let $(t_0, t_1), (t_2, t_3) \in B_m$ two distinct breaks and $S \subseteq O^M_{|m}$. The breaks $(t_0, t_1), (t_2, t_3)$ cannot simultaneously assigned to have value 1 in any feasible solution of $\mathcal{P}^B$ if*

$$\begin{aligned}&\Big|[t_0, t_1[_{\mathbb{Z}} \cap [\min_{(j,k)\in S}\{a_{j,k} - d^{se}_{j,k}\}, \max_{(j,k)\in S}\{f_{j,k} + d^{pr}_{j,k}\}[_{\mathbb{Z}}\Big| \\ &\quad + \Big|[t_2, t_3[_{\mathbb{Z}} \cap [\min_{(j,k)\in S}\{a_{j,k} - d^{se}_{j,k}\}, \max_{(j,k)\in S}\{f_{j,k} + d^{pr}_{j,k}\}[_{\mathbb{Z}}\Big| \\ &\quad > f^M_m - \Big(a^M_m + \sum_{(j,k)\in O^M_{|m}} (d^{pr}_{j,k} + d^{se}_{j,k})\Big).\end{aligned}$$

Then, the conflict

$$z^{rd,ru}_{m,t_0,t_1} + z^{rd,ru}_{m,t_2,t_3} \leq 1$$

is a valid constraint of $\mathcal{P}^B$.

Another valid class of conflicts is "conflicts between initial and final breaks on different machines". The idea is to fix an initial break $(t_0, t_1) \in B_m$ on machine m and to derive the resulting earliest possible final ramp down on machine $m_2 \in M$ by propagating the resulting processing starts of the tasks.

Conflicts Over Different Machines

The following condition describes a condition to derive a conflict between two breaks $(t_0, t_1) \in B_m$, $(t_2, t_3) \in B_{m_2}$ on machine $m, m_2 \in M$ that cannot be used simultaneously in any feasible solution of $\mathcal{P}^B$. The condition describes that the breaks cannot be used together when the break (t_0, t_1) delimits the processing start of a task (j, k) in such a way that the propagated completion time of a successor of task (j, k) conflicts with break (t_2, t_3).

Theorem 4.3.8. *Let $m, m_2 \in M$ two distinct machines and $(t_0, t_1) \in B_m$, $(t_2, t_3) \in B_{m_2}$ two breaks satisfying $t_1 < t_2$.*

Additionally let $j \in J$ be a job visiting m by task (j, k) before visiting m_2 by task (j, l). The break (t_0, t_1) and the break (t_2, t_3) cannot be used simultaneously in any feasible integer solution of $\mathcal{P}^B$ if the conditions

$$t_0 < a_{j,k}, \;\; \textit{and}\;\; t_1 + d^{se}_{j,k} + \sum_{l_3=k}^{l} d^{pr}_{j,l_3} > t_2, \;\; \textit{and}\;\; f_{j,l_3} \leq t_3$$

hold. Then, the conflict

$$z^{rd,ru}_{m,t_0,t_1} + z^{rd,ru}_{m_2,t_2,t_3} \leq 1$$

is a valid constraint of $\mathcal{P}^B$.

Proof. We are given the distinct machines $m, m_2 \in M$ and the breaks $(t_0, t_1) \in B_m$ and $(t_2, t_3) \in B_{m_2}$ satisfying $t_1 < t_2$. In addition, the job-sequence $j \in J$ starts processing of task (j, k) on machine m before the processing of task (j, l) on machine m_2. Moreover, the time windows of the tasks satisfy

$$t_0 < a_{j,k} + d^{pr}_{j,k} < f_{j,l} - d^{se}_{j,l} < t_3.$$

These conditions restrict the task (j, k) to start processing after the break (t_0, t_1). Additionally, the task (j, l) needs to start processing before break (t_2, t_3). Therefore, the complete processing of the tasks $(j, k), \dots, (j, l)$ needs to be completed within $[t_1 + d^{se}_{j,k}, t_2[_{\mathbb{Z}}$. The tasks $\{(j, k), \dots, (j, l)\}$ satisfy

$$t_1 + d^{se}_{j,k} + \sum_{l_3=k}^{l} d^{pr}_{j,l_3} > t_2.$$

Thus, the setup and the processing of the subset of the tasks $\{(j, k), \dots, (j, l)\}$ cannot be completed before period t_2, since the processing of the task (j, k) starts as early as possible but the task (j, l) cannot be finished in time. Therefore, the breaks (t_0, t_1) and (t_2, t_3) cannot be used within the same integer solution, as they prevent the complete processing of the tasks. Thus, at most, one of the breaks is allowed to be used in a feasible solution, and the conflict cut

$$z^{rd,ru}_{m,t_0,t_1} + z^{rd,ru}_{m_2,t_2,t_3} \leq 1$$

is valid for $\mathcal{P}^B$. □

This condition can be strengthened by also considering additional requirements, such as the processing and setup of further tasks. These conflicts can be detected by extensive probing, and we do not generate conflicts for pairs of breaks. Another possibility is to fix the processing start of additional tasks and analyze the forbidden processing starts.

However, these conflicts become ineffective for larger time windows. The schedule could be shifted in time arbitrarily. Moreover, the detection of these conflicts is time-consuming.

Conflicts Arising From Optimality Criteria

Describing conditions that must hold in optimal solutions can strengthen the LP-relaxation. This can be done by describing combinations of variables that will not be used simultaneously in optimal solutions. Thus, we now describe the more interesting criteria for conflicts. These are the conflicts that could be used to cut off branches early, e.g., prevent the generation of those branches. These conflicts are helpful when running into troubles caused by multiple near-optimal solutions.

Within the presolving and propagation rules, we mentioned the presolving by dominating sets, see Theorem 4.1.41. This rule describes that a break is redundant if we can substitute the break with a cheaper combination of breaks and standby. In contrast, if the break cannot be substituted by a cheaper sequence of breaks and standby, we can derive a conflict. Therefore, we use the mappings

$$bestcost : m \times [T^m_B[_{\mathbb{Z}} \times [T^m_B[_{\mathbb{Z}} \to \mathbb{R}$$

to compute the best objective value by standby and breaks on machine $m \in M$ and

$$bestchoice : \ m \times [T^m_B[_{\mathbb{Z}} \times [T^m_B[_{\mathbb{Z}} \to \mathcal{P}(B_m \cup [T[_{\mathbb{Z}}\ )$$

describing the corresponding subset of breaks and standby periods.

Theorem 4.3.9 (Conflicts by minimum distance of breaks)**.** *Let* $m \in M$ *be one machine. The breaks* $(t_0, t_1) \in B_m$ *and* $(t_2, t_3) \in B_m$, *with* $t_1 < t_2$, *will not be part of any optimal feasible integer solution simultaneously, if* $t_2 - t_1 < \min_{(j,k) \in O^M_{|m}} (d^{se}_{j,k} + d^{pr}_{j,k})$ *and*

$$\hat{d}_{m,t_0,t_1} + \hat{d}_{m,t_2,t_3} + bestcost(m, t_1, t_2) \geq \hat{d}_{m,t_0,t_3}$$

holds.

Proof. Let $m \in M$ a machine and $(t_0, t_1), (t_2, t_3) \in B_m$ two breaks on machine m satisfying $t_1 < t_2$. Moreover, the breaks satisfy

$$t_2 - t_1 < \min_{(j,k)\in O^M_{|m}} (d^{se}_{j,k} + d^{pr}_{j,k}).$$

The periods $t \in [t_1, t_2[_{\mathbb{Z}}$ cannot be used for setup or processing in a locally feasible integer solution. Thus, the periods $t \in [t_1, t_2[_{\mathbb{Z}}$ can only be covered by breaks or standby. Comparing the partial objective costs of (t_0, t_3) with the partial objective costs of a solution using the breaks (t_0, t_1), (t_2, t_3) leads to

$$\begin{aligned} \hat{d}_{m,t_0,t_1} + \hat{d}_{m,t_2,t_3} + bestcost(m, t_1, t_2) = & \\ & \hat{d}_{m,t_0,t_3} + bestcost(m, t_1, t_2) + \sum_{q=t_1-d^{ru}_m}^{t_1-1} D^{ru}_m P_q + \sum_{q=t_2}^{t_2+d^{rd}_m} D^{rd}_m P_q \\ & \geq \hat{d}_{m,t_0,t_3}. \end{aligned}$$

Thus, the solution $(t_0, t_1), (t_2, t_3)$ in combination with the best choice between t_1 and t_2 is not the optimal choice locally since the same schedule and a machine profile using break (t_0, t_3) describes a better objective value.

Suppose, there exists an integer feasible solution which uses

$$(t_0, t_1), bestchoice(m, t_2, t_3), (t_2, t_3).$$

Then, the solution can be improved by replacing (t_0, t_1), $bestchoice(m, t_1, t_2)$, (t_2, t_3) by (t_0, t_3). Thus, under the given circumstances, the use of (t_0, t_1) and (t_2, t_3) cannot lead to an optimal feasible integer solution respectively, there exists another optimal feasible integer solution that does not use (t_0, t_1) and (t_2, t_3) simultaneously. □

The graphical interpretation of this theorem is as shown in Figure 4.22.

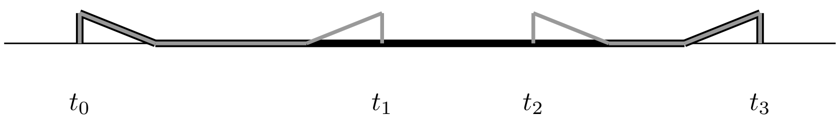


Figure 4.22: This figure shows three breaks: (t_0, t_1),(t_2, t_3) and (t_0, t_3). In addition, no task is allowed to start processing in any of the periods $[t_1, t_2[_{\mathbb{Z}}$. Then, concerning the mentioned constraints to the objective, the black break (t_0, t_3) is always the better choice for any integral solution than the usage of (t_0, t_1),(t_2, t_3) and some assignment of the periods $t \in [t_1, t_2[_{\mathbb{Z}}$ to break or standby.

These conflicts can be extended to conflicts of standby periods between a break and the processing start of a task. However, most of these complicated conflicts could in principle be found by extensive probing. However, this requires problem-specific probing implementation, which fixes the two breaks consecutively to 1.

4.3.2 Generalized Upper Bounds

Jorge P. Sousa and Laurence Wolsey proposed different valid generalized upper bounds (GUB) inequalities in [SW92] for time-indexed formulations of the single-machine scheduling problem.

Definition 4.3.10 (Generalized upper bound constraint)**.** *Let*

$$P = \operatorname{conv}(\{x \in \mathbb{R}^n \mid Ax \le b, \ x \in \{0,1\}^n\})$$

with $A \in \mathbb{Q}^{m\times n}$, $b \in \mathbb{Q}^m$ and $c \in \mathbb{Q}^n$ and $n, m \in \mathbb{N}$. For $J \subseteq \{1, \dots, n\}$ and $U \in \mathbb{Q}$, The constraint $\sum_{j\in J} x_j \le U$, is called a ***generalized upper bound constraint*** *for P if it describes a valid constraint for P.*

Wolsey applies the technique of GUB constraints on special knapsack problems in [Wol90], called GUB knapsacks. Those are knapsack problems with additional GUB constraints. The author mentions that the derived GUB cover constraints can strengthen the classical cover constraints. Wolsey and Sousa apply these techniques to the single-machine scheduling problem.

Among other things, van den Akker [vdA94] also analyzed GUB constraints in the case of single-machine scheduling. While Sousa and Wolsey [SW92] described valid GUB covers, van den Akker analyzed the properties of a class of GUB covers and proved that some are facet-defining.

Van den Akker et al. [AHS00] as well as Sousa and Wolsey [SW92] proposed classes of valid inequalities of a time-indexed formulation of the single-machine scheduling problem. Considering the single-machine scheduling problem, we are given $n \in \mathbb{N}$ jobs with processing durations $p_i \in \mathbb{N}$ that must be processed within a time window $[T[_\mathbb{Z}$. Note that van den Akker uses the notation $[a, b] = \{a + 1, \dots, b\}$. However, we present the results using our notation $[a, b[_\mathbb{Z} = \{a, \dots, b - 1\}$.

We will briefly reproduce the results of van den Akker and Wolsey and Sousa. Then, we will present a lifting scheme to adapt the inequalities to the job-shop scheduling problem setting with flexible energy prices and time windows.

Results in the Case of Single-Machine Scheduling

Van den Akker [vdA94], and also Sousa and Wolsey [SW92] consider the single-machine scheduling problem. A time-indexed formulation can describe the feasible solutions to the single-machine scheduling problem.

For simplification, we present the description of the single-machine scheduling problem and the valid inequalities and variables directly using our notation of the job-shop scheduling problem. Therefore, each job $j \in [n[_\mathbb{Z}$ of the single-machine scheduling is associated with a task $(j, k) = (j, 0)$ and the corresponding processing time p_j is equal to $d^{pr}_{j,k}$. The problem formulation is as follows:

$$
\begin{aligned}
\min \quad & c^\top x \\
s.t: \quad & \sum_{t \in [T - d^{pr}_{j,k}+1[_\mathbb{Z}} x_{j,k,t} = 1 && \forall\, (j,k) \in O^M_{|m} && (4.46)\\
& \sum_{(j,k)\in O^M_{|m}} \sum_{q=t-d^{pr}_{j,k}+1}^{t} x_{j,k,q} \le 1 && \forall\, t \in [T[_\mathbb{Z} && (4.47)\\
& x_{j,k,t} = 0 && \forall\, (j,k) \in O^M_{|m},\ t \in [T - d^{pr}_{j,k}+1, T[_\mathbb{Z} && (4.48)\\
& x_{j,k,t} \in \{0,1\} && \forall\, (j,k) \in O^M_{|m},\ t \in [T[_\mathbb{Z} && (4.49)
\end{aligned}
$$

Thereby, the set

$$
P^*_S = \operatorname{conv}\left(\left\{x \in \{0,1\}^{[T[_\mathbb{Z} \times \{1,\dots,n\}} \mid x \text{ satisfies (4.46) and (4.49)}\right\}\right)
$$

describes the polytope of the feasible solutions. Sousa and Wolsey aggregated the constraints 4.47 and analyzed the resulting problem formulation, consisting of $|O^M_{|m}|$ set packing constraints and one knapsack constraints:

$$
\begin{aligned}
U = \Big\{x \in \{0,1\}^{|O^M_{|m}| \times [T[_\mathbb{Z}} \mid\ & \sum_{(j,k)\in O^M_{|m}} \sum_{t=1}^{T-d^{pr}_{j,k}+1} a_{j,k,t} \cdot x_{j,k,t} \le b \\
& \sum_{t \in [T - d^{pr}_{j,k}[_\mathbb{Z}} x_{j,k,t} \le 1 \ \ \forall\, i \in \{1,\dots,n\} \\
& x_{j,k,t} = 0 \ \forall\, (j,k) \in O^M_{|m},\ t \in [T - d^{pr}_{j,k}+1, T[_\mathbb{Z} \\
& x_{j,k,t} \in \{0,1\} \ \forall\, (j,k) \in O^M_{|m},\ t \in [T[_\mathbb{Z} \Big\}.
\end{aligned}
$$

The polytopes P^*_S and U contain the same integral points, but U is described by a knapsack constraint, and P^*_S is described by set packing constraints. Instead of deriving cover-cuts, see [KNT98], from this knapsack constraint, the additional information of the GUB-constraint is used in order to derive stronger inequalities.

Definition 4.3.11. *The set $C \subseteq O^M_{|m} \times [T[_\mathbb{Z}$ is called a GUB cover for U, if the elements of C are pairwise distinct and $\sum_{(j,k,t)\in C} a_{j,k,t} > b$.*

With the set C, we associate the set $V(C) = \{j,k) \mid (j,k,t) \in C\}$ and the set of periods

$$
Q_{(j,k)} = \{t \mid a_{j,k,s} \ge a_{j,k,t},\ \text{where } (a_{j,k,t}) \in C\}
$$

and

$$Q'_{(j,k)} = \{s \in [T[_{\mathbb{Z}} \mid a_{j,k,s} \geq a_{i,l,t} \ \forall (i,l,t) \in C\}$$

for $(i,l) \in O^M_{|m} \setminus \in V(C)$.

Proposition 4.3.12 ([SW92]). *If C is a GUB cover for U, then*

$$\sum_{(j,k)\in V(C)} \sum_{q \in Q_{(j,k)}} x_{j,k,q} + \sum_{(j,k)\in O^M_{|m} \setminus V(C)} \sum_{q \in Q'_{(j,k)}} x_{j,k,q} \leq |C| - 1$$

is a valid inequality for U.

To obtain valid inequalities with right-hand-side 1, Sousa and Wolsey [SW92] propose to aggregate over Δ consecutive time periods $\{t, t+1, \ldots, t+\Delta-1\}$. Then, the resulting knapsack constraint can be described by the coefficients

$$a_{j,k,s} = \min\{d^{pr}_{j,k}, \Delta, (t+\Delta-s)^+, (s+d^{pr}_{j,k}-t)^+\}.$$

Let C be a GUB cover of size 2, then the sets $Q_{j,k}$ and $Q'_{j,k}$ result in

$$\begin{aligned} Q_{j,k} &= [t - d^{pr}_{j,k} - a_{j,k,t}, t + \Delta - a_{j,k,pr}[_{\mathbb{Z}} && \forall (j,k) \in V(C) \\ Q'_{j,k} &= [t - d^{pr}_{j,k} - \overline{a}, t + \Delta - \overline{a}[_{\mathbb{Z}} && \forall (j,k) \notin V(C), \end{aligned}$$

where $\overline{a} = \max\{a_{j,k,t} \mid (j,k) \in V(C)\}$.

Sousa and Wolsey provide the following classification of valid inequalities with right-hand side 1.

Theorem 4.3.13 ([SW92, Proposition 2]). *Consider a job (j,k), a period t, and $\Delta \in \{2, \ldots, \overline{p}\}$ where $\overline{p} = \max_{j \neq l}\{d^{pr}_{j,k}\}$. Then*

$$\sum_{s \in Q_{(j,k)}} x_{j,k,s} + \sum_{(i,l)\in O^M_{|m} \setminus \{(j,k)\}} \sum_{s \in Q'_{(i,l)}} x_{i,l,s} \leq 1$$

is a valid inequality of P^*_S, where $Q_{(j,k)} = [t - d^{pr}_{j,k} + 1, t + \Delta - 1[_{\mathbb{Z}}$, $Q'_{(j,k)} = [t - d^{pr}_{j,k} + \Delta, t[_{\mathbb{Z}}$ for $j \neq l$ such that $d^{pr}_{j,k} \geq \Delta$ and $Q'_{(j,k)} = \emptyset$ otherwise.

Van den Akker [vdAvHS99] complements and extends the results of Sousa and Wolsey [SW92] as follows. Among others, van den Akker derived the following results.

Definition 4.3.14. *We are given the optimization problem* $\min\{c^\top x \mid Ax \leq b, x \in \{0,1\}^n\}$ *with $c \in \mathbb{Q}^n, A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$, $m, n \in \mathbb{N}$. A constraint $x(V) = \sum_{q \in V} x_q \leq 1$ with $V \subseteq \{1, \ldots, n\}$ is called **maximal** if there is no valid constraint $x(W) \leq 1$ with $V \subset W \subseteq \{1, \ldots, n\}$.*

Definition 4.3.15. *Let $Q = \text{conv}(\{x \in \{0,1\}^n \mid Ax \leq b, x \in \{0,1\}^n\})$ be a polytope. A valid and maximal constraint $x(V) \leq 1$ for Q with $V \subset [n[_{\mathbb{Z}}$ is called facet-defining for Q, if*

$$dim(\{x \in Q \mid x(V) = 1\}) = dim(Q) - 1$$

holds.

Proposition 4.3.16 ([vdA94, Theorem 3.3]). *Any facet defining inequality $x(V) \leq 1$ for P^*_S with $V = V_{(j,k)} \cup (\bigcup_{(i,l)\in O^M_{|m} \setminus \{(j,k)\}} V_{(i,l)})$ has the following structure:*

$$\begin{aligned} V_{(j,k)} &= [l - d^{pr}_{j,k}, u[_{\mathbb{Z}} \quad \textit{and} \\ V_{(i,l)} &= [u - d^{pr}_{i,l}, l[_{\mathbb{Z}} && \textit{for all } (i,l) \in O^M_{|m} \setminus \{(j,k)\} \end{aligned}$$

with $l, u \in [T[_{\mathbb{Z}}$ and $l < u$.

Proposition 4.3.17. *A valid inequality $x(V) \leq 1$ is facet-defining for P^*_S if and only if the inequality is maximal.*

Corollary 4.3.18 ([vdA94, Theorem 3.4]). *A valid inequality $x(V) \leq 1$ with the structure described in Proposition 4.3.16 that is maximal is facet-defining for P^*_S.*

Van den Akker derived sufficient conditions for the maximality of a constraint $x(V) \leq 1$ in [vdA94].

Proposition 4.3.19 ([vdA94, Theorem 3.5]). *Let $(j,k) \in O^M_{|m}$ and $l, u \in [T[_{\mathbb{Z}}$ with $l < u$ and $V = V_{(j,k)} \cup (\bigcup_{(i,l) \in O^M_{|m} \setminus \{(j,k)\}} V_{(i,l)})$. An inequality $x(V) \leq 1$ is maximal, if it has the structure described in Proposition 4.3.16 and $V_{(i,l)} \neq \emptyset$ for at least one $(i,l) \in O^M_{|m} \setminus \{(j,k)\}$.*

The mentioned GUB cover constraints and the analysis of GUB cover constraints were only considered for pre-schedules. A pre-schedule is a scheduling of the tasks, which need not schedule each task.

Van den Akker also extends in [vdA94] the theory of GUB cover constraints in the case of single-machine scheduling to conditions whether the derived constraint is also facet-defining for the single-machine scheduling, where each task must be processed. However, we do not present nor recreate those parts within this thesis since our problem setting also includes more complex substructures. The following sections will add problem structures to the GUB cover constraints. We will start with the additional consideration of setup times. To that end, we provide a linear transformation of the processing and setup times of the tasks to show that setup times can be considered by the GUB cover cuts. Finally, we show that the constraints are facet-defining inequalities under special circumstances of a subproblem of the job-shop scheduling problem with flexible energy prices and time windows. We focus on the job-shop scheduling problem with flexible energy prices and time windows, restricted to a single machine $m \in M$. This single-machine problem is a subproblem of the considered job-shop scheduling problem with flexible energy prices and time windows. To that end, the derived valid inequalities of this single-machine scheduling subproblem are valid inequalities of the job-shop scheduling problem with flexible energy prices and time windows.

GUB Covers for Single-Machine Scheduling with Setup Times

To consider setup times, we analyze a subproblem of the job-shop scheduling problem with flexible energy prices and time windows, namely the subproblem of scheduling the tasks on a single machine $m \in M$ with flexible energy prices and time windows.

We start with the single-machine scheduling subproblem and extend this problem by considering setup times. Each task can be processed within the time window, but the setup must be completed directly before processing. The following polytope P^*_{setup} describes all feasible pre-schedules:

$$
\begin{aligned}
P^*_{setup} = \operatorname{conv}\Bigg(\Bigg\{ x \in \{0,1\}^{O^M_{|m} \times [T[_{\mathbb{Z}}} \;\Bigg|\; & \\
& \sum_{t=d^{se}_{j,k}}^{T-d^{pr}_{j,k}+1} x_{j,k,t} \leq 1 && \forall\ (j,k) \in O^M_{|m} \\
& \sum_{(j,k) \in O^M_{|m}} \sum_{s=t-d^{pr}_{j,k}+1}^{t+d^{se}_{j,k}} x_{j,k,t} \leq 1 && \forall\ t \in [T[_{\mathbb{Z}} \\
& x_{j,k,t} = 0 && \forall\ (j,k) \in O^M_{|m},\ t \in [T - d'_{j,k} T[_{\mathbb{Z}} \Bigg\}\Bigg).
\end{aligned}
$$

We introduce a new processing duration

$$
\hat{d}^{pr}_{j,k} = d^{se}_{j,k} + d^{pr}_{j,k},
$$

for each task $(j,k) \in O^M_{|m}$ by combining the original processing and setup durations. The instance and the corresponding problem formulation need to be reformulated using the new processing times. We introduce new variables $\hat{x}_{j,k,t}$ for each $(j,k) \in O^M_{|m}$ and $t \in [T[_{\mathbb{Z}}$ which are equal to one, if task (j,k) starts processing (with processing duration $\hat{d}^{pr}_{j,k}$) in period t and zero otherwise. Therefore, we get a single-machine scheduling problem with time window T and $|O^M_{|m}|$ tasks with processing times $\hat{d}^{pr}_{j,k}$ and the theory of van den Akker, and of Sousa and Wolsey can be applied.

Given an instance of the single machine job-shop scheduling problem and a solution for the $\hat{x}$-variables, the corresponding solution for the x can be retrieved by the linear transformation $x_{j,k,t} = \hat{x}_{j,k,t-d^{se}_{j,k}}$ for $t \in [d^{se}_{j,k}, T[_{\mathbb{Z}}$ and $x_{j,k,t} = 0$ for $t \in [d^{se}_{j,k}[_{\mathbb{Z}}$. Thus, we can also consider the polytope

$$
\begin{aligned}
\hat{P}_{setup^*} = \text{conv}\Bigg(\Bigg\{\hat{x} \in \{0,1\}^{O^M_{|m} \times [T[_{\mathbb{Z}}} \;\Bigg|\; & \\
& \sum_{t=0}^{T-\hat{d}^{pr}_{j,k}+1} \hat{x}_{j,k,t} \le 1 && \forall\ (j,k) \in O^M_{|m} \\
& \sum_{(j,k)\in O^M_{|m}} \sum_{s=t-\hat{d}^{pr}_{j,k}+1}^{t} \hat{x}_{j,k,t} \le 1 && \forall\ t \in [T[_{\mathbb{Z}} \\
& x_{j,k,t} = 0 && \forall\ (j,k) \in O^M_{|m},\ t \in [T - \hat{d}_{j,k} T[_{\mathbb{Z}} \Bigg\}\Bigg).
\end{aligned}
$$

The inequality

$$\sum_{t\in V_{(j,k)}} \hat{x}_{j,k,t} + \sum_{(i,l)\in O^M_{|m}\setminus\{(j,k)\}} \sum_{t\in V_{(i,l)}} \hat{x}_{i,l,t} \le 1 \tag{4.50}$$

with $l, u \in [T[_{\mathbb{Z}}$, $l < u$, and

$$
\begin{aligned}
V_{(j,k)} &= [l - \hat{d}^{pr}_{j,k} + 1, u + 1[_{\mathbb{Z}} \quad \text{and} \\
V_{(i,l)} &= [u - \hat{d}^{pr}_{i,l} + 1, l + 1[_{\mathbb{Z}} && \forall\ (i,l) \in O^M_{|m}
\end{aligned}
$$

is valid for $\hat{P}^*_{setup}$, since $\hat{P}^*_{setup}$ is described by a time-indexed formulation of a single-machine scheduling problem. Applying the linear (re-)transformation on the $\hat{x}$-variables lead to the constraint

$$\sum_{t\in V_{(j,k)}} x_{j,k,t+d^{se}_{i,l}} + \sum_{(i,l)\in O^M_{|m}\setminus\{(j,k)\}} \sum_{t\in V_{(i,l)}} x_{i,l,t+d^{se}_{i,l}} \le 1 \tag{4.51}$$

Inequality (4.51) is valid since it is derived from a valid inequality of the single-machine scheduling problem by a linear transformation. The index shift in constraint (4.52) can be directly considered within the sets $V_{j,k}$.

Now we reproduce results from [vdA94] with the additional consideration of setup times.

Corollary 4.3.20. *Let $(j,k) \in O^M_{|m}$ be one task on machine m. In addition let $l, u \in [T[_{\mathbb{Z}}$. Then, for*

$$
\begin{aligned}
V_{(j,k)} &= [l - d^{pr}_{j,k} + 1, u + d^{se}_{j,k} + 1[_{\mathbb{Z}} \\
\textit{and } V_{(i,l)} &= [u - \hat{d}^{pr}_{i,l} + 1, l + d^{se}_{i,l} + 1[_{\mathbb{Z}} && \forall\ (i,l) \in O^M_{|m}
\end{aligned}
$$

the constraint

$$\sum_{t\in V_{(j,k)}} x_{j,k,t} + \sum_{(i,l)\in O^M_{|m}\setminus\{(j,k)\}} \sum_{t\in V_{(i,l)}} x_{i,l,t} \le 1 \tag{4.52}$$

is valid for P^*_{setup}.

Theorem 4.3.21. *If the inequality (4.52) is maximal, then the inequality is facet-defining for P^*_{setup}.*

Proof. As mentioned before, for $(j,k) \in O^M_{|m}$, the constraint

$$\sum_{t\in V_{(j,k)}} \hat{x}_{j,k,t} + \sum_{(i,l)\in O^M_{|m}\setminus\{(j,k)\}} \sum_{t\in V_{(i,l)}} \hat{x}_{i,l,t} \le 1$$

is facet-defining for $\hat{P}^*_{setup}$ with

$$
\begin{aligned}
V_{(j,k)} &= [l - \hat{d}^{pr}_{j,k} + 1, u + 1[_{\mathbb{Z}} \\
\text{and } V_{(i,l)} &= [u - \hat{d}^{pr}_{i,l} + 1, l + 1[_{\mathbb{Z}} && \forall\ (i,l) \in O^M_{|m}
\end{aligned}
$$

if the constraint is maximal. The mapping

$$\phi : P^*_{setup} \to \hat{P}^*_{setup}, x_{j,k,t} \mapsto \begin{cases} \hat{x}_{j,k,t-d^{se}_{j,k}} & \text{if } t - d^{se}_{j,k} \ge 0 \\ \hat{x}_{j,k,T-d^{se}_{j,k}+t} & else \end{cases}$$

is an isomorphism. Since $dim(P^*_{setup}) = dim(\hat{P}^*_{setup})$, the property of the constraints is also mapped from $\hat{P}^*_{setup}$ to the polytope P^*_{setup}. Thus, the constraint (4.52) is facet-defining for P^*_{setup}, if Constraint (4.50) is maximal. □

Extending the single-machine scheduling problem by setup times requires a linear transformation.

Corollary 4.3.22. *Let* $(j,k) \in O^M_{|m}$ *be a task and* $l, u \in [T[_{\mathbb{Z}}$ *two distinct periods. The set* $V = V_{(j,k)} \cup (\bigcup_{(i,l)\in O^M_{|m}\setminus\{(j,k)\}} V_{(i,l)})$ *satisfies*

$$\begin{aligned} V_{(j,k)} &= [l - \hat{d}^{pr}_{j,k} + 1, u + 1[_{\mathbb{Z}} \\ \text{and } V_{(i,l)} &= [u - \hat{d}^{pr}_{i,l} + 1, l + 1[_{\mathbb{Z}} \qquad\qquad \forall\ (i,l) \in O^M_{|m}. \end{aligned}$$

Then, the inequality (4.52) *is maximal, if* $V_{(i,l)} \neq \emptyset$ *holds for one task* $(i,l) \in O^M_{|m} \setminus \{(j,k)\}$.

GUB Cover Constraints Considering Breaks

Now, we consider the single-machine scheduling problem setup times and breaks, formulated by the following polytope

$$\begin{aligned} P^*_{break} = \operatorname{conv}\Bigg(\bigg\{ & x \in \{0,1\}^{O^M_{|m}\times[T[_{\mathbb{Z}}}\ ,\ z^{rd,ru}_{m,t_0,t_1} \in \{0,1\} \quad \forall (t_0,t_1) \in B_m \\ & \sum_{t=d^{se}_{j,k}}^{T-d^{pr+1}_{j,k}} x_{j,k,t} \leq 1 \quad \forall (j,k) \in O^M_{|m} \\ & \sum_{(t_0,t_1)\in B_m : t\in[t_0,t_1[_{\mathbb{Z}}} z^{rd,ru}_{m,t_0,t_1} + \sum_{(j,k)\in O^M_{|m}} \sum_{s=t-d^{pr}_{j,k}+1}^{t+d^{se}_{j,k}} x_{j,k,t} \leq 1 \quad \forall\ t \in [T[_{\mathbb{Z}} \\ & x_{j,k,t} = 0 \quad \forall\ (j,k) \in O^M_{|m},\ t \in [T - \hat{d}_{j,k} T[_{\mathbb{Z}} \bigg\}\Bigg). \end{aligned}$$

Remark 4.3.23. *The feasible solutions of* P^*_{setup}*, extended by* 0 *for each break* $(t_0,t_1) \in B_m$*, are feasible solutions of* P^*_{break}*. Thus, Constraint* (4.52) *is also a valid constraint of* P^*_{break} *for each* $l, u \in [T[_{\mathbb{Z}}$ *and* $(j,k) \in O^M_{|m}$.

Clearly, the constraints (4.52) describe facets of

$$F = \Big\{(x, z^{rd,ru}) \in P^*_{break} \mid z^{rd,ru}_{m,t_0,t_1} = 0 \quad \forall\ (t_0,t_1) \in B_m\Big\},$$

if the constraint is facet-defining for P^*_{setup}.

Now, the idea is to lift each break variable $z^{rd,ru}_{m,t_0,t_1}$, for $(t_0,t_1) \in B_m$, into the constraints (4.52). Lifting increases the dimension of a valid inequality by adding variables. Iteratively, the variables are added to the constraints. The corresponding coefficient is chosen as large as possible while maintaining the validity of the new inequality.

We are using the Theorem 4.3.24 to describe the lifting procedure.

Theorem 4.3.24 ([WN14], p.261 Proposition 1.1)**.** *For* $S = \operatorname{conv}\{\chi(M) \mid M \subseteq \{1,\dots,n\}\}$ *we define* $S^\delta := S \cap \{x \in S : x_1 = \delta\}$ *for* $\delta \in \{0,1\}$. *Suppose*

$$\sum_{j=2}^{n} \pi_j u_j \leq \pi_0$$

is valid for S^0. *If* $S^1 = \emptyset$*, then* $x_1 \leq 0$ *is valid for* S. *If* $S^1 \neq \emptyset$*, then*

$$\alpha_1 x_1 + \sum_{j=2}^{n} \pi_j u_j \leq \pi_0$$

is valid for S *for any* $\alpha_1 \leq \pi_0 - \max\{\sum_{j=2}^n \pi_j x_j \mid x \in S^1\}$.

Moreover, if

$$\alpha_1 = \pi_0 - \max\Big\{\sum_{j=2}^{n} \pi_j x_j \mid x \in S\Big\} \text{ and } \sum_{j=2}^{n} \pi_j x_j \leq \pi_0$$

gives a face of dimension k *of* S^0*, then* $\alpha_1 + \sum_{j=2}^n \pi_j x_j \leq \pi_0$ *gives a face of dimension at least* $k+1$ *of* S.

If $\sum_{j=2}^n \pi_j x_j \leq \pi_0$ *is a facet of* S^0*, then*

$$\alpha_1 + \sum_{j=2}^{n} \pi_j x_j \leq \pi_0$$

is a facet of S *for* $\alpha_1 = \pi_0 - \max\{\sum_{j=2}^n \pi_j x_j \mid x \in S\}$.

Now, we will use Theorem 4.3.24 to add the break variables to constraint (4.52) with their maximal coefficients.

Theorem 4.3.25. *Let $l, u \in [T[_{\mathbb{Z}}$ two periods and $(j,k) \in O^M_{|m}$ a task processed by machine $m \in M$. The break $(t_0, t_1) \in B_m$ can be lifted into constraints* (4.52) *with coefficient* 1*, iff the condition*

$$t_0 \le l < u \le t_1$$

holds.

Proof. The constraint

$$\sum_{t \in V_{(j,k)}} x_{j,k,t} + \sum_{(i,l) \in O^M_{|m} \setminus \{(j,k)\}} \sum_{t \in V_{(i,l)}} x_{i,l,t} \le 1$$

is valid for ${P^*_{break}}^0 = \{(x, z^{rd,ru}) \in P^*_{break} \mid z^{rd,ru}_{m,t_0,t_1} = 0 \ \forall\ (t_0, t_1) \in B_m\}$. Now, we are able to lift the break (t_0, t_1) into the constraint (4.52) with coefficient

$$\alpha_{m,t_0,t_1} = 1 - \max\Big\{ \sum_{t \in V_{(j,k)}} x_{j,k,t} + \sum_{(i,l) \in O^M_{|m} \setminus \{(j,k)\}} \sum_{t \in V_{(i,l)}} x_{i,l,t} : \\ (x, z^{rd,ru}) \in P^*_{break}, \quad z^{rd,ru}_{m,t_0,t_1} = 1 \Big\}.$$

Let $(t_0, t_1) \in B_m$ satisfy $t_0 \le l < u \le t_1$. The processing of task (j,k) cannot start in period $l - d^{pr}_{j,k}$, since $l - d^{pr}_{j,k} + 1 + d^{pr}_{j,k} \ge t_0$ would lead to a conflict of the processing of task (j,k) and the break (t_0, t_1). Analogously, the task cannot start in any of the periods $t \in [l - d^{pr}_{j,k} + 1, u + d^{se}_{j,k} + 1[_{\mathbb{Z}}$. The same argumentation is valid for task $(i,l) \in O^M_{|m} \setminus \{(i,l)\}$ and processing starts in $[u - d^{pr}_{i,l} + 1, l + d^{se}_{i,l} + 1[_{\mathbb{Z}}$. Thus, for each $(x, z^{rd,ru}) \in P^*_{break}$

$$\begin{aligned} x_{i,l,t} &= 0 && \forall\ t \in [u - d^{pr}_{i,l} + 1, l + d^{se}_{i,l} + 1[_{\mathbb{Z}} \\ \text{and } x_{j,k,t} &= 0 && \forall\ t \in [l - d^{pr}_{j,k} + 1, u + d^{se}_{j,k} + 1[_{\mathbb{Z}} \end{aligned}$$

holds.

Therefore, the maximal value of $\sum_{t \in V_{(j,k)}} x_{j,k,t} + \sum_{(i,l) \in O^M_{|m} \setminus \{(j,k)\}} \sum_{t \in V_{(i,l)}} x_{i,l,t}$ equals 0, if $t_0 \le l < u \le t_1$ holds. Thus, the maximum coefficient α_{m,t_0,t_1} of break (t_0, t_1) is 1. To show that this scheme lifts all breaks to its maximal coefficient, we need that this scheme does not miss a break.

Suppose the break $(t_0, t_1) \in B_m$ satisfies $t_0 \le l \le t_1 < u$. Then, the optimal solution to the maximization problem

$$\max\Big\{ \sum_{t \in V_{(j,k)}} x_{j,k,t} + \sum_{(i,l) \in O^M_{|m} \setminus \{(j,k)\}} \sum_{t \in V_{(i,l)}} x_{i,l,t} : (x, z^{rd,ru}) \in P^*_{break}, \quad z^{rd,ru}_{m,t_0,t_1} = 1 \Big\} = 0$$

since the task (j,k) can start processing in period $u - 1 + d^{se}_{j,k} \in V_{(j,k)}$.

In the case of $l < t_0 \le u \le t_1$, the task (j,k) can start processing in period $l \in V_{(j,k)}$. The case $l < t_0 \le t_1 < u$ allows the same argumentation. Thus, the case $t_0 \le l < u \le t_1$ is the only case where the maximal coefficient is 1.

Therefore, the break (t_0, t_1) can be lifted into Constraint (4.52) with maximum coefficient 1, if and only if $t_0 \le l \le u \le t_1$ holds and with coefficient 0 otherwise. □

The next theorem shows that all breaks can be lifted into the constraint iteratively with coefficient 1 if they satisfy a certain condition. Moreover, the theorem states that the lifting scheme is sequence-independent. This property is not present for general lifting schemes because various sequences of lifting the variables into the constraints can result in different valid constraints for the given polytope. In the case of the considered constraint class, the lifting sequence can be chosen arbitrarily if the lifting function is superadditive [GNS00]. However, we can show that each break that can be lifted with coefficient 1 into the constraint conflicts with all the other breaks with coefficient 1, and thus, the lifting function can neglect breaks.

Proposition 4.3.26. *Let $m \in M$, $(j,k) \in O^M_{|m}$ and $l, u \in [T[_{\mathbb{Z}}$. Let $S \subseteq B_m$ a subset of breaks satisfying*

$$t_2 \le l < u \le t_3 \ \forall\ (t_2, t_3) \in S.$$

Then, the constraint

$$\sum_{t \in V_{(j,k)}} x_{j,k,t} + \sum_{(i,l) \in O^M_{|m} \setminus \{(j,k)\}} \sum_{t \in V_{(i,l)}} x_{i,l,t} + \sum_{(t_2,t_3) \in S} z^{rd,ru}_{m,t_2,t_3} \le 1.$$

is valid for P^*_{break} and the lifting order is sequence-independent.

Proof. Let $(j,k) \in O^M_{|m}$ and $l,u \in [T[_{\mathbb{Z}}$. Additionally, let $S \subseteq B_m$ a subset of breaks satisfying

$$t_2 \leq l < u \leq t_3 \ \forall \ (t_2,t_3) \in S.$$

Then, each break $(t_2,t_3) \in S$ can be lifted into constraint (4.52).

Let $S_0 \subset S$ and $(t_0,t_1) \in S \setminus S_0$. Then, the maximal coefficient of break (t_0,t_1) can be computed by

$$\alpha_{m,t_0,t_1} = \max\Big\{1 - (\sum_{t\in V_{(j,k)}} x_{j,k,t} + \sum_{(i,l)\in O^M_{|m}\setminus\{(j,k)\}} \sum_{t\in V_{(i,l)}} x_{i,l,t} + \sum_{(t_2,t_3)\in S_0} z^{rd,ru}_{m,t_2,t_3}) \Big| \ (x, z^{rd,ru}) \in P^*_{break}, \ z^{rd,ru}_{m,t_0,t_1} = 1\Big\}.$$

If $t_0 \leq l < u \leq t_1$ holds, then the constraint

$$\sum_{(t_2,t_3)\in S_0} z^{rd,ru}_{m,t_2,t_3} \leq 1 - z^{rd,ru}_{m,t_0,t_1}$$

is valid for P^*_{break} since each break, which is part of $S_0 \cup \{(t_0,t_1)\}$, covers interval $[l,u[_{\mathbb{Z}}$. However, only one break is allowed to cover this interval within a feasible solution. Thus, the sum $\sum_{(t_2,t_3)\in S_0} z^{rd,ru}_{m,t_2,t_3}$ can be neglected within the optimization problem since it is fixed to 0. Thus, the simplified objective of the coefficient optimization problem is

$$1 - \Big(\sum_{t\in V_{(j,k)}} x_{j,k,t} + \sum_{(i,l)\in O^M_{|m}\setminus\{(j,k)\}} \sum_{t\in V_{(i,l)}} x_{i,l,t}\Big)$$

and the coefficient of (t_0,t_1) can be computed by solving the lifting-problem for constraint (4.52). Thus, the lifting-coefficient of break (t_0,t_1) is independent from S_0. Therefore, the subset of already lifted breaks does not affect the coefficient of break (t_0,t_1), and the lifting procedure is sequence-independent. The feasibility of the constraint

$$\sum_{t\in V_{(j,k)}} x_{j,k,t} + \sum_{(i,l)\in O^M_{|m}\setminus\{(j,k)\}} \sum_{t\in V_{(i,l)}} x_{i,l,t} + \sum_{(t_2,t_3)\in S} z^{rd,ru}_{m,t_2,t_3} \leq 1.$$

is derived from the validity of the lifting scheme.

□

Proposition 4.3.27. *Let $(j,k) \in O^M_{|m}$ and $l,u \in [T[_{\mathbb{Z}}$. If the constraint*

$$\sum_{t\in V_{(j,k)}} x_{j,k,t} + \sum_{(i,l)\in O^M_{|m}\setminus\{(j,k)\}} \sum_{t\in V_{(i,l)}} x_{i,l,t} \leq 1$$

is a maximal constraint of P^*_{setup}, then the constraint

$$\sum_{t\in V_{(j,k)}} x_{j,k,t} + \sum_{(i,l)\in O^M_{|m}\setminus\{(j,k)\}} \sum_{t\in V_{(i,l)}} x_{i,l,t} + \sum_{(t_0,t_1)\in B_m : t_0\leq l\leq u\leq t_1} z^{rd,ru}_{m,t_0,t_1} \leq 1 \tag{4.53}$$

is facet-defining for P^*_{break}.

GUB Cover Constraints Considering Standby Variables

To additionally consider the standby variables when describing GUB cover constraints of single-machine scheduling with setup times, breaks and standby, we need to analyze the following problem description:

$$\begin{aligned} P^*_{all} = \operatorname{conv}\Big(\Big\{& x \in \{0,1\}^{O^M_{|m}\times[T[_{\mathbb{Z}}}, \ z^{rd,ru}_{m,t_0,t_1} \in \{0,1\} \quad \forall (t_0,t_1) \in B_m, z^{st} \in \{0,1\}^{[T[_{\mathbb{Z}}} \\ & \sum_{t=d^{se}_{j,k}}^{T-d^{pr+1}_{j,k}} x_{j,k,t} \leq 1 \quad \forall (j,k) \in O^M_{|m}, \\ & \sum_{(t_0,t_1)\in B_m : t\in[t_0,t_1[_{\mathbb{Z}}} z^{rd,ru}_{m,t_0,t_1} + \sum_{(j,k)\in O^M_{|m}} \sum_{s=t-d^{pr}_{j,k}+1}^{t+d^{se}_{j,k}} x_{j,k,t} + z^{st}_{m,t} \leq 1 \quad \forall t \in [T[_{\mathbb{Z}} \ , \\ & x_{j,k,t} = 0 \quad \forall \ (j,k) \in O^M_{|m}, \ t \in [T - \hat{d}_{j,k} T[_{\mathbb{Z}} \ \Big\}\Big). \end{aligned}$$

Proposition 4.3.28. *Let $l, u \in [T[_{\mathbb{Z}}$ and $(j,k) \in O^M_{|m}$. Then, the inequality*

$$\sum_{t \in V_{(j,k)}} x_{j,k,t} + \sum_{(i,l) \in O^M_{|m} \setminus \{(j,k)\}} \sum_{t \in V_{(i,l)}} x_{i,l,t} + \sum_{(t_0,t_1) \in B_m : t_0 \leq l \leq u \leq t_1} z^{rd,ru}_{m,t_0,t_1} \leq 1$$

is a valid constraint for P^*_{all}.

The validity of Proposition 4.3.28 follows from the fact that P^*_{break} describes the feasible solutions of P^*_{all}, if $z^{st}_{m,t} = 0$ is fixed for all each $m \in M$ and $t \in [T[_{\mathbb{Z}}$. Let $l, u, t \in [T[_{\mathbb{Z}}$ with $l + 1 < u$. Now we consider the lifting expression for standby variable $z^{st}_{m,t}$:

$$\begin{aligned} \alpha_{m,t} = \max \Big\{ 1.0 - \Big(& \sum_{t \in V_{(j,k)}} x_{j,k,t} + \\ & \sum_{(i,l) \in O^M_{|m} \setminus \{(j,k)\}} \sum_{t \in V_{(i,l)}} x_{i,l,t} + \sum_{(t_0,t_1) \in B_m : t_0 \leq l \leq u \leq t_1} z^{rd,ru}_{m,t_0,t_1} \Big) \Big| \\ & z^{st}_{m,t} = 1 \\ & (x, z^{rd,ru}, z^{st}) \in P^*_{all} \Big\}. \end{aligned}$$

If $t < l$ or $t \geq u$ hold, then the objective value of this optimization problem is 0 since the standby fixation does not affect the maximum left-hand side of the constraint.

If $l \leq t \leq u - 1$ holds, then the value of $\sum_{(t_0,t_1) \in B_m : t_0 \leq l \leq u \leq t_1} z^{rd,ru}_{m,t_0,t_1} = 0$, since

$$\sum_{(t_0,t_1) \in B_m : t_0 \leq l \leq u \leq t_1} z^{rd,ru}_{m,t_0,t_1} \leq \sum_{(t_0,t_1) \in B_m : t \in [t_0,t_1[_{\mathbb{Z}}} z^{rd,ru}_{m,t_0,t_1} = 0$$

must hold. The maximum value is bounded by

$$\sum_{t \in V_{(j,k)}} x_{j,k,t} + \sum_{(i,l) \in O^M_{|m} \setminus \{(j,k)\}} \sum_{t \in V_{(i,l)}} x_{i,l,t} \leq 1$$

since (4.52) is a valid constraint of P^*_{all}. The optimization problem attains its maximum value of 0 because there is always a period $q \in V_{j,k}$ such that $\sum_{t \in V_{(j,k)}} x_{j,k,t} = 1$ and $z^{st}_{m,t} = 1$. Thus, the standby variables can be lifted into the constraints (4.53) and (4.52) with coefficient 0, if $l < u - 1$. In the case of $t = l$ and $l + 1 = u$, the derived GUB cover is the already existing constraint.

$$\sum_{(t_0,t_1) \in B_m : t \in [t_0,t_1[_{\mathbb{Z}}} z^{rd,ru}_{m,t_0,t_1} + \sum_{(j,k) \in O^M_{|m}} \sum_{s = t - d^{pr}_{j,k} + 1}^{t + d^{se}_{j,k}} x_{j,k,t} + z^{st}_{m,t} \leq 1.$$

Proposition 4.3.29. *Let $m \in M$ and $l, u \in [T[_{\mathbb{Z}}$ with $l < u - 1$. Then, the constraint*

$$\sum_{t \in V_{(j,k)}} x_{j,k,t} + \sum_{(i,l) \in O^M_{|m} \setminus \{(j,k)\}} \sum_{t \in V_{(i,l)}} x_{i,l,t} + \sum_{(t_0,t_1) \in B_m : t_0 \leq l \leq u \leq t_1} z^{rd,ru}_{m,t_0,t_1} \leq 1$$

is facet-defining for P^*_{all} if the constraint

$$\sum_{t \in V_{(j,k)}} x_{j,k,t} + \sum_{(i,l) \in O^M_{|m} \setminus \{(j,k)\}} \sum_{t \in V_{(i,l)}} x_{i,l,t} \leq 1$$

is a maximal constraint for P^*_{setup}.

The facet-defining property is always coupled to the maximality of the GUB cover constraint for P^*_{setup}.

Proposition 4.3.30. *If the* (4.52) *is maximal for P^{setup}, then the lifted constraint* (4.53) *is maximal for P^{all} for $l < u$.*

Proposition 4.3.31 (Theorem 3.5 [vdA94]). *Let $m \in M$ be one machine. The constraint* (4.52) *is maximal for $l, u \in [T[_{\mathbb{Z}}$ with $[l, u[_{\mathbb{Z}} \neq [T[_{\mathbb{Z}}$ and $(j,k) \in O^M_{|m}$, if $V_j \neq \emptyset$ for one $(i,l) \in O^M_{|m} \setminus \{(j,k)\}$.*

Since P^*_{all} describes a relaxation of a subproblem of the job-shop scheduling problem with flexible energy prices and time windows, the validity of the derived constraints is also given for $\mathcal{P}^B$. Nevertheless, even the maximality condition is not guaranteed in the case of $\mathcal{P}^B$ since we also need to consider time windows and precedence constraints. Thus, future work will require further attempts to derive conditions to maximality in combination with time windows and precedence constraints.

GUB Covers With Right-Hand-Side ≥ 2

The presented GUB cover constraints are constraints with right-hand-side 1. Sousa and Wolsey also present valid inequalities with right-hand-side $|I|$ for a subset $I \subseteq O^M_{|m}$ in the case of single-machine scheduling. The setup times of the tasks can be considered equal to what we did for the cover cuts with right-hand-side 1.

Proposition 4.3.32 ([SW92],Proposition 5). *Let $m \in M$ and $I \subset O^M_{|m}$ a non-empty subset of tasks. Let*

$$\Delta = \sum_{(j,k)\in I} d^{\hat{pr}}_{j,k} + \delta$$

with $\delta \in [\max_{(j,k)\in I}\{d^{pr}_{j,k} + d^{se}_{j,k}\} - 1, \max_{(j,k)\in O^M_{|m}}\{d^{pr}_{j,k} + d^{se}_{j,k} - 1\} + 1[_{\mathbb{Z}}$. Additionally, there is the second set of tasks $I^\delta = \{(i,l) \in O^M_{|m} \mid d^{pr}_{i,l} + d^{se}_{i,l} \geq \delta + 1\} \setminus I$. Then the inequality

$$\sum_{(j,k)\in I} \sum_{q=t}^{t+\Delta-d^{pr}_{j,k}+d^{se}_{j,k}} x_{j,k,q} + \sum_{(i,l)\in I^\delta} \sum_{q=t+\delta+1-d^{pr}_{i,l}}^{t+\Delta-\delta-1+d^{se}_{i,l}} x_{i,l,q} \leq |I| \tag{4.54}$$

is valid for P^{setup}.

The validity of this proposition can be verified by the usage of the combined setup and processing times $\hat{d}^{pr}_{j,k}$ for each task $(j,k) \in O^M_{|m}$ in case of the single-machine scheduling and the validity of the GUB cover constraints in [SW92, Proposition 5]. The constraint (4.54) is also a valid constraint of P^*_{all}. To additionally consider breaks within this constraint, a lifting scheme is necessary.

Theorem 4.3.33. *Consider the machine $m \in M$. For $I \subset O^M_{|m}$ and $t \in [T[_{\mathbb{Z}}$, and $\delta \in [\max_{(j,k)\in I}\{d^{pr}_{j,k} + d^{se}_{j,k}\} - 1, \max_{(j,k)\in O^M_{|m}}\{d^{pr}_{j,k} + d^{se}_{j,k} - 1\} + 1[_{\mathbb{Z}}$ we are given the constraint*

$$\sum_{(j,k)\in I} \sum_{q=t}^{t+\Delta-d^{pr}_{j,k}+d^{se}_{j,k}} x_{j,k,q} + \sum_{(i,l)\in I^\delta} \sum_{q=t+\delta+1-d^{pr}_{i,l}}^{t+\Delta-\delta-1+d^{se}_{i,l}} x_{i,l,q} \leq |I|.$$

The break variable $z^{rd,ru}_{m,t_0,t_1}$ of break $(t_0,t_1) \in B_m$ can be lifted into this constraints, if

$$|[t_0,t_1[_{\mathbb{Z}} \cap [t+1, t+\Delta-\delta-1[_{\mathbb{Z}}| \geq \delta + 1$$

holds.

Proof. For $m \in M$, we are given the non-empty subset of tasks $I \subset O^M_{|m}$ and $t \in [T[_{\mathbb{Z}}$. Then, with $\delta \in \{\max_{(j,k)\in I}\{d^{pr}_{j,k}+d^{se}_{j,k}\}-1, \ldots, \max_{(j,k)\in O^M_{|m}}\{d^{pr}_{j,k}+d^{se}_{j,k}-1\}\}$ we are given the constraint:

$$\sum_{(j,k)\in I} \sum_{q=t}^{t+\Delta-d^{pr}_{j,k}+d^{se}_{j,k}} x_{j,k,q} + \sum_{(i,l)\in I^\delta} \sum_{q=t+\delta+1-d^{pr}_{i,l}}^{t+\Delta-\delta-1+d^{se}_{i,l}} x_{i,l,q} \leq |I|. \tag{4.55}$$

This constraint describes the fact that within the interval of length $\Delta + \delta$ periods, only a limited number of tasks from the set I and from I^δ can be processed. If the break claims more than $\delta + 1$ periods of the interval $[t+1, t+\Delta-\delta-1[_{\mathbb{Z}}$, then the break prevents at least one task $(j,k) \in I$ from starting processing within the considered interval.

To determine the lifting coefficient, we use the lifting theorem 4.3.24. The constraint (4.54) is valid, if we fix the break variables $z^{rd,ru}_{m,t_0,t_1}$ to zero.

The maximum coefficient of break (t_0,t_1) can be computed by $\alpha_{m,t_0,t_1} = |I| - \xi$ with

$$\xi = \max\{\sum_{(j,k)\in I} \sum_{q=t}^{t+\Delta-d^{pr}_{j,k}+d^{se}_{j,k}} x_{j,k,q} + \sum_{(i,l)\in I^\delta} \sum_{q=t+\delta+1-d^{pr}_{i,l}}^{t+\Delta-\delta-1+d^{se}_{i,l}} x_{i,l,q} \mid x \in P^*_{break},\ z^{rd,ru}_{m,t_0,t_1} = 1\}.$$

The break (t_0,t_1) can be lifted into the constraints with positive coefficients if the fixation of $z^{rd,ru}_{m,t_0,t_1} := 1$ prevents at least one task $(j,k) \in I$ from starting processing in $[t + d^{se}_{j,k}, t + \Delta - \delta + 1[_{\mathbb{Z}}$. Therefore, we treat the break (t_0,t_1) as a task, starting processing in period t_0 with a processing duration $t_1 - t_0$. Therefore, this break would be part of I^δ since the notional processing duration satisfies $t_1 - t_0 \geq \delta + 1$. If a task of I^δ starts processing within the considered interval, then one task (j,k) in I cannot start processing. Therefore, the parameter α_{m,t_0,t_1} satisfies $\alpha_{m,t_0,t_1} = 1$ holds. $\square$

Algorithm 4 Greedy Algorithm For Set I

Require: Machine m, set of operations $O^M_{|m}$, fractional processing starts $\hat{x}_{j,k,t}$

Choose $I \subset O^M_{|m}$, $\|I\| = 2$
repeat
  $I_{best} = \emptyset$
  $val = 0$
  for $(j,k) \in O^M_{|m} \setminus I$ **do**
    if $val < f(I \cup \{(j,k)\}) - f(I_{best})$ **then**
      $val = f(I \cup \{(j,k)\}) - f(I)$
      $I_{best} = I \cup \{(j,k)\}$
    end if
  end for
  $I = I_{best}$
until $I_{best} = \emptyset$
return I_{best}

The class of possible constraints of type (4.54) is exponential-sized. To efficiently compute violated constraints, we use a greedy heuristic to detect violated constraints. The objective function $f : \mathcal{P}(O^M_{|m}) \to \mathbb{R}$ considered in the greedy algorithm is defined as

$$f(I) = \max\{ \sum_{(j,k)\in I} \sum_{q=t}^{t+\Delta-d^{pr}_{j,k}+d^{se}_{j,k}} x_{j,k,q} + \sum_{(i,l)\in I^\delta} \sum_{q=t+\delta+1-d^{pr}_{i,l}}^{t+\Delta-\delta-1+d^{se}_{i,l}} x_{i,l,q} - |I| \\ \mid t \in [T[_{\mathbb{Z}}\ ,\ \delta, \Delta, I^\delta \text{as above}\}.$$

For each $m \in M$, the Greedy algorithm tries to compute a set $I \subseteq O^M_{|m}$, such that the corresponding constraint (4.54) is violated for at least one $t \in [T[_{\mathbb{Z}}$. The Greedy algorithm iteratively extends a set I until no element $(j,k) \in O^M_{|m} \setminus I$ exists, which increases the objective value. One evaluation of $f(I)$ for an arbitrary $I \subset O^M_{|m}$ requires at most $\mathcal{O}(|O^M_{|m}| \cdot T)$ operations. The evaluation of the objective of the maximization problem is necessary once for each δ using efficient incremental summation algorithms. Thus, the evaluation for each $t \in [T[_{\mathbb{Z}}$ leads to a total number of $\mathcal{O}(T^2 \cdot |O^M_{|m}|)$ operations.

Moreover, Sousa and Wolsey proposed a lifting scheme to increase the value of the coefficients of the task variables. The lifting scheme in [SW92] is described in the following theorem. Within our implementation, we are using the following lifting scheme. The lifting scheme is similar to [SW92], but the processing times are considered to equal the setup and the processing durations.

Theorem 4.3.34 (Lifting scheme)**.** *Let $m \in M$ one machine. The tasks $(j_c,k_c),(j_r,k_r) \in O^M_{|m}$ are index in such that $d^{se}_{j_c,k_c} + d^{pr}_{j_c,k_c} \leq d^{se}_{j_r,k_r} d^{pr}_{j_r,k_r}$ if $c < r$ holds. Then, the coefficient of $x_{i,l,s}$ can be lifted to α, if*

$$s \in \left[t + \sum_{c=l-c+2}^{l} d^{se}_{j_c,k_c} + d^{pr}_{j_c,k_c} + \delta + 1 - d^{se}_{i,l} - d^{pr}_{i,l}, t + \sum_{c=l-c+2}^{l} d^{se}_{j_c,k_c} + d^{pr}_{j_c,k_c} - 1\right] \neq \emptyset$$

for $(i,l) \in O^M_{|m}$ and $s \in [T[_{\mathbb{Z}}$.

Sousa and Wolsey provide the proof in case of the transformed processing times. The validity when considering setup times follows directly by considering the combined processing duration $\hat{d}^{pr}_{j,k}$ for each $(j,k) \in O^M_{|m}$.

4.3.3 Valid Constraints From Linear Ordering

This section considers the linear ordering subproblem of scheduling formulations. As described in Section 3.2.4, the linear ordering formulation coupled with the time-indexed variables needs many coupling precedence constraints. Moreover, ordering tasks on a single machine is not a dual-bound driving aspect of the solution process. Therefore, only violated constraints should be considered when solving the LP-relaxations.

We present valid inequalities derived from the linear ordering problem formulation that could improve the problem description. The idea is to take two distinct tasks $(j,k),(i,l) \in$

$O^M_{|m}$ and to combine the associated constraints

$$\sum_{q=0}^{t-d^{pr}_{j,k}-d^{se}_{i,l}} x_{j,k,q} - \sum_{q=0}^{t} x_{i,l,q} \geq (p^{i,l}_{j,k} - 1) \ \forall\ t \in [T[_{\mathbb{Z}} \tag{4.56}$$

and

$$p^{i,l}_{j,k} + p^{j,k}_{i,l} \geq 1 \tag{4.57}$$

$$p^{i,l}_{j,k} + p^{j,k}_{i,l} \leq 1 \tag{4.58}$$

to eliminate the $p^{i,l}_{j,k}$ variables. Reordering the constraints leads to the following description for $p^{i,l}_{j,k}$:

$$1 + \sum_{q=0}^{t-d^{pr}_{j,k}-d^{se}_{i,l}} x_{j,k,q} - \sum_{q=0}^{t} x_{i,l,q} \geq p^{i,l}_{j,k} \ \forall\ t\ \in [T[_{\mathbb{Z}}$$

Let $(j,k),(i,l) \in O^M_{|m}$, $(j,k) \neq (i,l)$, two distinct tasks one machine $m \in M$.

Substituting the expression in (4.56) with periods $t, t_2 \in [T[_{\mathbb{Z}}$, in (4.57) leads to the following expression:

$$\sum_{q=0}^{t-d^{pr}_{j,k}-d^{se}_{i,l}} x_{j,k,q} - \sum_{q=0}^{t} x_{i,l,q} + \sum_{q=0}^{t_2-d^{pr}_{i,l}-d^{se}_{j,k}} x_{i,l,q} - \sum_{q=0}^{t_2} x_{j,k,q} \geq -1\ .$$

Scaling the inequality by -1 and rearranging the sums leads to

$$\sum_{q=0}^{t_2} x_{j,k,q} - \sum_{q=0}^{t-d^{pr}_{j,k}-d^{se}_{i,l}} x_{j,k,q} + \sum_{q=0}^{t} x_{i,l,q} - \sum_{q=0}^{t_2-d^{pr}_{i,l}-d^{se}_{j,k}} x_{i,l,q} \leq 1\ .$$

Since the problem formulation already includes the constraints

$$\sum_{t\in[T[_{\mathbb{Z}}} x_{j,k,t} = 1$$

$$\text{and} \sum_{t\in[T[_{\mathbb{Z}}} x_{i,l,t} = 1$$

we only create new constraints if $t_2 > t - d^{pr}_{j,k} - d^{se}_{i,l}$ and $t > t_2 - d^{pr}_{i,l} - d^{se}_{j,k}$ hold simultaneously.

Then, the constraints, derived from the linear ordering subproblem, can be formulated for $(j,k),(i,l) \in O^M_{|m}$, $m \in M$ and particular choices $t, t_2 \in [T[_{\mathbb{Z}}$ by

$$\sum_{q=t-d^{pr}_{j,k}-d^{se}_{i,l}+1}^{t_2} x_{j,k,q} + \sum_{q=t_2-d^{pr}_{i,l}-d^{se}_{j,k}+1}^{t} x_{i,l,q} \leq 1. \tag{4.59}$$

In the case of $t_2 = t - d^{pr}_{j,k} - d^{se}_{i,l} + 1 \in [T[_{\mathbb{Z}}$, we get the constraints

$$x_{j,k,t_2} + \sum_{q=t_2-d^{pr}_{i,l}-d^{se}_{j,k}+1}^{t_2+d^{pr}_{j,k}+d^{se}_{i,l}-1} x_{i,l,q} \leq 1. \tag{4.60}$$

These constraints describe the arising conflicts of a processing start of task (j,k) in period t_2 and the allowed processing starts of task (i,l).

Proposition 4.3.35. *Let $m \in M$ and $(j,k),(i,l) \in O^M_{|m}$, $(j,k) \neq (i,l)$. For the periods $t, t_2 \in [T[_{\mathbb{Z}}$, the constraints*

$$\sum_{q=t-d^{pr}_{j,k}-d^{se}_{i,l}+1}^{t_2} x_{j,k,q} + \sum_{q=t_2-d^{pr}_{i,l}-d^{se}_{j,k}+1}^{t} x_{i,l,q} \leq 1.$$

is valid for $\mathcal{P}^B$.

In general, Constraint (4.59) can be extended by further tasks and breaks, for example, breaks covering all used processing starts.

Some experiments indicate that the constraints (4.59) are not sufficiently effective in driving the dual bound in a significant manner. This is additionally supported by the fact that the optimal execution order does not lead to integer solutions in general, see Section 4.2.6. We only put the constraints (4.60) into the conflict graph.

Proposition 4.3.36. *Let $m \in M$ and $(j,k),(i,l) \in O^M_{|m}$, $(j,k) \neq (i,l)$. The conflicts* (3.10b) *and* (4.60) *are sufficient to represent the inequalities* (4.59) *in the conflict graph.*

By usage of constraint (3.10b) and constraints (4.60), the conflict described by (4.59) can be formulated.

4.4 Column Generation

A well-known approach in integer linear programming is to treat a subset of variables implicitly if the number of variables is too large [BJN+98]. This approach, called column generation, solves the linear programming relaxation at the branch-and-bound nodes by generating the columns only at the moment when they need to enter the LP. As mentioned, the sets of breaks B_m, for each $m \in M$, become large but do not grow exponentially. Nevertheless, in a near-optimal integer feasible solution, the number of used breaks would not exceed the number of tasks by much since one will not use two consecutive breaks between two processings in case of meaningful energy prices. Thus, a large subset of all breaks will not used in near-optimal integer feasible solutions, and the generation and usage within the model would be unnecessary.

This section is organized as follows: first, the pricing problem structure for break variables is introduced. Next, a straightforward pricing algorithm is introduced. The presentation of an efficient pricing algorithm is followed by the analysis of the trivial algorithm. In the case of this algorithm, the consideration of the propagation and presolving rules will be discussed. In addition, the consideration of different cutting planes is described.

As for every $m \in M$, B_m is large, and many breaks are not applicable. Thus, we use a branch-and-price approach [BJN+98] to solve our model. A huge number of break variables is redundant. Additionally, many variables cannot participate in any feasible integer solution because using the break variable conflicts with the required task processing on the same machine. To overcome the problem of generating and including redundant variables, we generate these variables only if they can be useful and if they can improve the LP solution. We present a condition to recognize the useful break variables from the not-applicable ones. To generate a break variable with negative reduced costs that can improve the LP solution, we present a *node-weighted shortest-problem*, which can be extended to a *hop-constraint node-weighted shortest-path problem*.

Starting with a restricted master problem (RMP) using only a subset $\hat{B}_m \subseteq B_m$ of all break variables for each machine $m \in M$. Then, we iteratively (re-)solve the RMP and add missing variables $z^{rd,ru}_{m,t_0,t_1}$ with $(t_0,t_1) \in B_m \setminus \hat{B}_m$ with negative reduced costs when solving the LP-relaxation of (3.10a)–(3.10i). We choose $\hat{B}_m := \{(-d^{rd}_m, d^{ru}_m), (T - d^{rd}_m, T + d^{ru}_m)\}$ for the initial set of break variables, as this guarantees the feasibility of the restricted model if there exists any feasible solution for the given instance. Those breaks correspond to the break variables implying the earliest ramp-up and the latest ramp-down. In the case of the root LP, if no feasible solution exists, including the initial branching variables, then there cannot be any feasible solution. For an LP at a node within the branch-and-bound tree, branching decisions may result in infeasible LPs, which can be resolved using Farkas pricing.

The pricing problem for the break variables $z^{rd,ru}_{m,t_0,t_1}$, $(t_0,t_1) \in B_m$, can be formulated and solved as a *node-weighted shortest-path problem* in the time-expanded machine state network for each machine $m \in M$ individually. Thereby, the underlying graph is acyclic. For each tuple (s,t), $s \in \{\mathit{off}, \mathit{ru}, \mathit{rd}\}$ and $t \in T^m_B$, we introduce a node (s,t). Additionally, there is a an arc between the nodes $(s_1,t_1),(s_2,t_2)$ if and only if one can switch from state s_1 to s_2 in exactly $t_2 - t_1$ periods. We also add artificial *start*- and *end*-nodes connected to the *ramp-down* and *ramp-up*-nodes as illustrated in Figure 4.23 (with an additional row to describe the period of the columns). We denote the graph of a given instance as $G_m = (V, A)$ with $V = \{(s,t) \mid s \in \mathtt{S} \text{ and } t \in T_B\}$. The set of arcs A is described above.

Visiting the node (s,t) describes that the machine must be in state s within a certain period of time: either one period, if $s = \mathit{off}$ holds, or the number of ramping periods of the specific ramping state. This means that the dual variables of the relevant inequalities of the periods and the associated energy prices and energy consumption must be summed up.

Combining the dual variables $\pi_{m,t}$ of constraints (3.10e), (3.10f) and (3.10d) to node

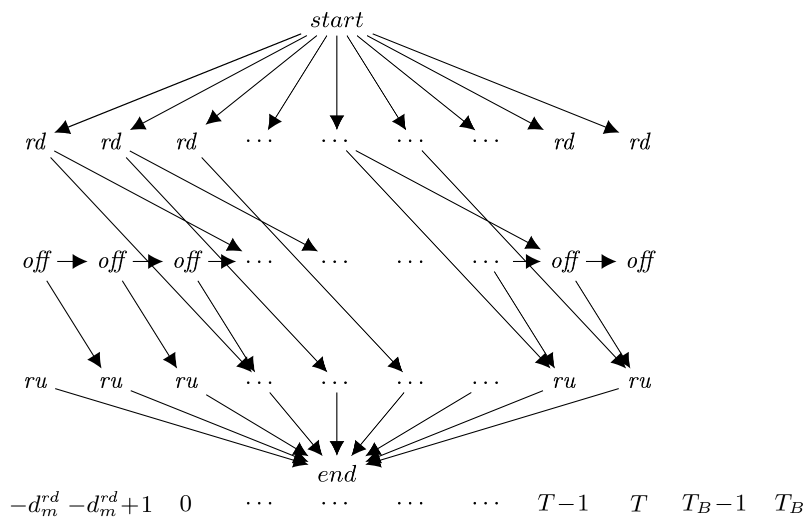


Figure 4.23: Time-expanded machine state network used in pricing problem.

weights ℓ as follows

$$\begin{aligned}
\ell_{start} &= \ell_{end} = 0, \\
\ell_{(off,t)} &= -\pi_{m,t} \quad \forall\, t \in T_B \\
&\text{and} \\
\ell_{(s,t)} &= \sum_{q=t}^{t+d_m^s-1} (C_q D_m^s - \pi_{m,q}) \text{ for } s \in \{ru, rd\},
\end{aligned}$$

leads to the following statement about a path from *start* to *end* in this graph. To recap, the energy costs C_t for $t \in T_B \setminus [T]$ are zero. Any path

$$start - (rd, t_0) - (off, t_1) - \cdots - (off, t_{n-1}) - (ru, t_n) - end$$

from *start* to an *end*-node describes a break variable, representing a feasible ramp-down, $n-1$ times of offline periods and a ramping up.
The proposed break variable on machine $m \in M$ denotes the following states:

- the ramping-down starts in period t_0 and ends in period $t_1 - 1$
- the machine is in state *off* from period t_1 to period t_{n-1}
- the machine starts a ramping-up in period t_n
- the machine can be in a setup, processing, standby-state, or using another break variable in period $t_n + d_m^{ru}$.

By this argumentation, any path from *start* to any *end* node describes a possible break variable describing a reasonable ramping on the machine.

4.4.1 Solving the Pricing Problem With a Shortest Path Algorithm

The theory of shortest-path problems and their variations offer a large set of possible algorithms. We only need to devise an acyclic shortest path algorithm, described by Algorithm 5: The runtime of the proposed Algorithm 5 is $\mathcal{O}(|V| + |T_B^m| \cdot |A|) = \mathcal{O}(|T_B^m|^2)$ since each node is watched exactly one time and each and the distance update of each node is done at most B_m^p times. One crucial aspect in Algorithm 5 is the existence of the

Algorithm 5 Acyclic shortest path with node weights

```
procedure ShortestPath(m, D = (V, A))
    dist_v = ∞   ∀ v ∈ V \ {start} ,
    set prec_v = −1   ∀ v ∈ V \ {start} and dist_{start,0} = 0
    create topological {v_1, ..., v_|V|} order of all vertices
    for (s_1, t_1) = v ∈ {v_1, ..., v_|V|} do
        for ((s_1, t_1), (s_2, t_2)) ∈ δ⁺(v) do
            if dist_(s_1,t_1) + ℓ_(s,t) < dist_(s_2,t_2) then
                dist_(s_2,t_2) = dist_(s_1,t_1) + ℓ_(s_2,t_2)
                prec_(s_2,t_2) = (s_1, t_1)
            end if
        end for
    end for
    Reconstruct the shortest path p
    return p, dist_end
end procedure
```

topological order of the vertices. This order is given by

$$\begin{aligned} v_1 &= start \\ v_2 &= (rd, -d_m^{rd}) \\ v_3 &= (rd, -d_m^{rd} + 1) \\ &\vdots \\ v_{|V|-1} &= (ru, T) \\ v_{|V|} &= end. \end{aligned}$$

This leads to a topological order because the predecessors of each node have smaller keys, and the process starts with the node with the smallest key.
Algorithm 5 computes a break with the smallest reduced costs. Since the presented graph G_m is acyclic, the algorithm always converges with a single break corresponding to a path.

4.4.2 A Hop-Constrained Shortest Path Problem

In Section 4.1, different techniques to detect redundant and non-usable breaks are presented. Considering these techniques within the pricing subproblem is a valid approach. However, the eliminated variables can be generated again within the pricing algorithm, although the propagation algorithm eliminates them. To overcome this problem, the pricing algorithm needs to get the information on whether variables are eliminated. A globally valid bound to the length of any break $(t_0, t_1) \in B_m$ on machine $m \in M$ is given by

$$t_1 - t_0 \leq T - \sum_{(j,k) \in O_{|m}} d_{j,k}^{pr} + d_{j,k}^{se}.$$

This expression describes that the maximal length of a break variable cannot exceed the remaining number of periods on machine $m \in M$ if all tasks are processed directly one after another. The validity of this bound was analyzed in 4.15. Within this subsection, we want to present an algorithm considering these bounds. To consider the restriction on the length of the breaks, we need to devise a hop constraint shortest path algorithm, which computes a shortest path with a limited number of arcs. The transformation of the bound on periods B_P into a bound on hops B_H on machine m can be done by

$$B_m^P - d_m^{ru} - d_m^{|rd} + 2 + 1 + 1 = B_m^H.$$

The bound considers one hop for ramping up and one hop for ramping down, one hop from the start node, and one hop to the end node.

The presented hop constraint shortest path algorithm is an acyclic shortest path algorithm within a hop expand network. The complexity of this algorithm is $\mathcal{O}(|T_B^m|^3)$.

The Algorithm 6 computes the reduced costs of the breaks with the smallest reduced costs, ending in period t_1 with length l for all $t_1 \in [T[_{\mathbb{Z}}$ and $l \in B_m^H$. The presolving conditions for breaks are presented in Section 4.1. Algorithm 6 computes a list of possible breaks

Algorithm 6 Hop constrained acyclic shortest path with node weights

```
procedure ShortestPath (m, D = (V, A))
    dist_{v,l} = ∞   ∀ v ∈ V \ {start}, l ∈ T_B ,
    set prec_{v,l} = −1   ∀ v ∈ V \ {start}, l ∈ T_B and dist_{start,0} = 0
    create topological {v_1, ..., v_{|V|}} order of all vertices v ∈ V
    for (s_1, t_1) = v ∈ {v_1, ..., v_{|V|}} do
        for l, l + 1 ∈ [B^H_m] do
            for ((s_1, t_1), (s_2, t_2)) ∈ δ^+(v) do
                if dist_{(s_1,t_1),l} + ℓ_{(s,t)} < dist_{(s_2,t_2),l+1} then
                    dist_{(s_2,t_2),l+1} = dist_{(s_1,t_1),l} + ℓ_{(s_2,t_2)}
                    prec_{(s_2,t_2,l+1)} = (s_1, t_1, l)
                end if
            end for
        end for
    end for
    Compute minimum with a check of usefulness
                ▷ Evaluation of presolving conditions while computing the minimum
    Rebuild the shortest path p
    return p, dist_{end}
end procedure
```

with negative reduced costs. A second iteration through the list can identify breaks that could be eliminated by applying the presolving conditions outlined in Section 4.1. However, many rules are too time-consuming, especially the bin-packing presolving technique 4.31g. Therefore, we present rules to simplify this presolving condition.

A break (t_0, t_1) is locally invalid, if there exists one task $(j, k) \in O^M_{|m}$ on machine $m \in M$, such that

$$\min(f_{j,k} - d^{se}_{j,k} + 1, t_1) - \max(a_{j,k} + d^{pr}_{j,k} - 1, t_0) > f_{j,k} - a_{j,k} - (d^{pr}_{j,k} + d^{se}_{j,k}).$$

This condition was already discussed in Section 4.1 in Theorem 4.13.

A similar condition can be derived for two distinct tasks $(j, k), (i, l) \in O^M_{|m}$ on machine $m \in M$, which are coupled by a precedence constraint $(j, k) \to (i, l)$. This case leads to the following intervals:

$$I_{right} = [a_{j,k} + d^{pr}_{j,k} + d^{se}_{i,l} + d^{pr}_{i,l} - 1, \ldots, f_{i,l} - d^{se}_{i,l} + 1],$$
$$I_{left} = [a_{j,k} + d^{pr}_{j,k} - 1, \ldots, f_{i,l} - d^{se}_{i,l} - d^{se}_{j,k} + d^{pr}_{j,k} + 1].$$

If both intervals I_{left}, I_{right} are covered by the break at hand, then we can conclude that this break cannot be used in a locally feasible integer solution. This can be verified as follows. Let $I_{right} \subseteq [t_0, t_1[_{\mathbb{Z}}$ with $(t_0, t_1) \in B_m$. Suppose a feasible integer solution uses the break (t_0, t_1). Then, the tasks (j, k) and (i, l) cannot be processed directly one after the other since $t_0 < a_{j_1,k_1} + d^{pr}_{j_1,k_1} + d^{se}_{j_2,k_2} + d^{pr}_{j_2,k_2} - 1$, and $f_{i,l} - d^{se}_{i,l} + 1 \leq t_1$. Suppose the break (t_0, t_1) can be placed between the tasks. Then, the processing and setup can be finished between a_{j_1,k_1} and t_0. But the setup of task (j_2, k_2) cannot start in time to start processing before period f_{j_2,k_2}, since $t_1 + d^{>}_{j_2,k_2} f_{j_2,k_2} + 1$ hold. Thus, the processing of both tasks cannot be completed in each possible constellation. The check, whether a presolving condition is satisfied or not, is summarized by **check of usefulness**. The rejection of break variables because of not passing the check of usefulness still leads to a description of all feasible integer solutions since we only discard variables that never participate in a feasible solution in the branch-and-bound subtree. The validity of this discarding procedure is proven in Section 4.1.

The hop-constrained node-weighted shortest path problem can improve the LP relaxation since the description of all integer feasible solutions becomes tighter by reducing the set of all possible break B_m to the subset of all useful break variables.

4.4.3 Fast Enumeration of All Break Variables

The pricing algorithm 6 has a runtime of $\mathcal{O}(|T^m_B|^3)$. Therefore, a substructure of the breaks is analyzed, and the substructure is exploited to speed-up the computation of the reduced costs.

Remark 4.4.1. *Let $m \in M$ be one machine and $(t_0, t_1), (t_0, t_1 + 1) \in B_m$ two distinct breaks. Then, the objective coefficient of $(t_0, t_1 + 1)$ can be computed from the objective coefficient of (t_0, t_1) by the formula*

$$\hat{d}_{m,t_0,t_1+1} = \hat{d}_{m,t_0,t_1} - C_{t_1 - d_m^{ru}} D_m^{ru} + C_{t_1} D_m^{ru}.$$

The reduced costs of a break $(t_0, t_0 + l) \in B_m$ can be computed by

$$red_{t_0,l} = \sum_{q=t}^{t+d_m^{rd}-1} \left(C_q D_m^{rd} - \pi_{m,q} \right) - \sum_{q=t+d_m^{rd}}^{t+l-d_m^{ru}-1} \pi_{m,q} + \sum_{q=t+l-d_m^{ru}}^{t_0+l-1} \left(C_q D_m^{ru} - \pi_{m,q} \right).$$

A path in the considered network using the arc $(start, t_0)$ can only vary the number of offline periods.

The underlying network is an acyclic digraph, and the structure of the break variables is always the same. This implies that initially, on machine $m \in M$, there are d_m^{rd} periods dedicated to ramping down, followed by an almost arbitrary number of offline periods, and finally, d_m^{ru} periods for ramping up. If we keep the start t_0 of the break, we can only vary the number of offline periods. The period t_1 of the end of the ramping-up results by the number of offline periods and the start of the break variable.

Using the idea of Observation 4.4.1 leads to the following update formula for the reduced costs:

$$\begin{aligned} red_{t_0,l+1} &= red_{t_0,l} - \left(C_{t_0+l} D_m^{ru} - \pi_{m,t+l-d_m^{ru}} \right) - \pi_{m,t+l+1} + \left(C_q D_m^{ru} - \pi_{m,t+l+1} \right) \\ &= red_{t_0,l} - C_{t_0+l} D_m^{ru} + \left(C_q D_m^{ru} - \pi_{m,t+l+1} \right). \end{aligned} \tag{4.61}$$

The update formula (4.61) describes that the reduced costs of a break with length l, ending in period $t_1 - 1$, can be computed from the reduced costs of the break variable with length $l - 1$, ending in period $t - 1$, by changing the costs to be the costs of an offline period and adding the costs of a ramp-up period. This computation has to be done for ever $t \in T_B$ and every $l \in T_B$ with $t \geq l \geq d_m^{rd} + d_m^{rd}$.

The update formula requires initial computation of the $\mathcal{O}(|T_B^m|)$ many reduced costs for the breaks each break $(t_0, t_0 + d_m^{rd} + d_m^{ru}) \in B_m$. Afterward, the update formulation can be used to compute the remaining breaks. Thus, the algorithm equals a brute-force enumeration of all breaks. However, the update formula leads to a speed-up compared to the Algorithm 6. The efficient enumeration algorithm requires $|T_B^m| \cdot |T_B^m|$ many operations to compute the initial set of reduced costs. Then, the $|T_B^m|$ many initial reduced costs are extended within $2 \cdot |T_B^m|$ steps. Thus, the efficient brute force enumeration algorithm requires $\mathcal{O}(|T_B^m|^2)$ many operations.

Variants of Iterating Over the Machines

Solving the restricted master problem requires several LP iterations of solving the master problem and the subproblems iteratively until no more variables with negative reduced costs are found. Instead of iterating over the machines by taking the machine subproblem in the order as the machines are stored in the list M, we propose sorting the subproblems in order of the sum of the negative reduced costs. This leads to the following statement: the subproblem with a high probability of returning a break variable with negative reduced costs is solved before the subproblem with a low probability. Other rules are to solve the machines in a given order and sort the subproblems according to the number of priced variables.

GUB Cover Constraints and Pricing

Separated GUB cover constraints (4.53) can be considered within the pricing problem. There could exist a GUB cover constraint for each task $(j,k) \in O_{|m}^M$ and $l, u \in T_B^m$.

Denote $\beta_{m,(j,k),l,u}$ the dual coefficient of constraint (4.53) for $(j,k) \in O_{|m}^M$ and $l, u \in T_B^m$. Then, the reduced costs of the break variable $z_{m,t_0,t_1}^{rd,ru}$ are determined by

$$\begin{aligned} &\sum_{q=t}^{t+d_m^{rd}-1} \left(C_q D_m^{rd} - \pi_{m,q} \right) - \sum_{q=t+d_m^{rd}}^{t+l-d_m^{ru}-1} \pi_{m,q} + \sum_{q=t+l-d_m^{ru}}^{t_0+l-1} \left(C_q D_m^{ru} - \pi_{m,q} \right) \\ &\qquad + \sum_{(j,k) \in O_{|m}^M} \sum_{l,u \in [T[_{\mathbb{Z}} \,:\, t_0 \leq l \leq u \leq t_1} \beta_{m,(j,k),l,u}. \end{aligned}$$

The dual coefficients of (4.53) need only be considered within the computation of the reduced costs of break $(t_0, t_1) \in B_m$, if $t_0 \leq < u \leq t_1$ holds. The term

$$\sum_{(j,k) \in O^M_{|m}} \sum_{l,u \in [T[_{\mathbb{Z}} \,:\, t_0 \leq l \leq u \leq t_1} \beta_{m,(j,k),l,u}$$

depends on the start and the length of the break variable. The sum of the dual coefficients must be computed for each possible break afterward. The dual coefficients cannot be considered in our time-expanded network representation, since these reduced costs of a certain constraint (4.53) is only relevant for a certain number of paths (breaks) within the time-expanded network.

The enumerative approach can consider this reduced cost. Within the reduced cost update step within the algorithm, the starting period t_0 and the length of the break are known. Thus, all constraints are known, where the break $(t_0, t_0 + l)$ participates. Also, if the break (t_0, t_1) participates in the GUB cover constraint of task $(j, k) \in O^M_{|m}$ and $l, u \in T^m_B$, then also the break $(t_0, t_1 + 1)$ participates in the same GUB cover constraint. Thus, the enumerative approach can also extend the reduced costs iteratively with the dual coefficients of the GUB cover constraints. The detailed consideration of these constraints within a branch-and-price approach leads to many challenges when it comes to the efficient handling of cutting planes and the management of separated inequalities. The implementation of the necessary data structures would be complex. Therefore, we are satisfied at this point with the indication of the possibility of implementation.

4.5 Primal algorithms and Heuristics

When solving MILPs by branch-and-bound, heuristics are crucial. Initial primal solutions improve the solving process in many ways. Solutions derived by heuristics are used to prune branches of the branch-and-bound tree where no improving feasible solutions are located. Therefore, near-optimal solutions are preferred in the early stages of the branch-and-bound process.

Further techniques, for example, cutting planes and propagation schemes, benefit from near-optimal primal bounds. These techniques could use the bound to derive valid constraints, consider the objective value, or fix variables that cannot obtain other values than in the optimal solution. An example is the reduced costs fixing technique presented in [BS15].

This section introduces the implemented heuristics for generating feasible primal solutions for the job-shop scheduling problem with flexible energy prices and time windows.

4.5.1 Heuristics in MILP-Solvers

Commercial MILP solvers are mostly used as black-box solvers. Different combinatorial algorithms are implemented within these solvers: algorithms to detect disconnected subproblems and special problem structures and inequalities. Nevertheless, black-box solvers need to be able to solve problems of realistic sizes. In addition, expensive heuristics, exploiting specific problem structures, are typically not implemented or used very often. Therefore, commercial solvers must provide heuristics applicable to many problems, and we have to add the problem-specific algorithms ourselves.

This section describes the implemented heuristics. Most of the considered heuristics are variants of classical scheduling heuristics.

4.5.2 List Scheduling

List scheduling algorithms are Greedy-algorithms and are mentioned, for example, in the books [Pin08, WS11]. A list scheduling algorithm schedules the tasks in a predefined execution order onto the machines as early as possible. The definition of the execution order of the tasks can vary, but in the end, the tasks need to be within an ordered list. The different ordering rules lead to different approximation ratios for minimizing the makespan. Since we consider a job-shop scheduling problem, each task must be processed by a predefined machine. The remaining flexibility is given in the period of the processing starts. In contrast to examples where the list scheduling heuristic provides an approximation ratio, we must consider precedence constraints and time windows.

The Algorithm 7 is a forward list scheduling algorithm that also considers the precedence constraints of the job sequences, the machine assignment and the time windows of the tasks. This algorithm is called "forward list scheduling" since the scheduling process starts at the beginning of the time window and ends at the end of the time window and goes forward in the processing times in time. The list scheduling algorithm starts with an

Algorithm 7 List scheduling for JSS

```
procedure LIST SCHEDULING HEURISTIC(Instance 𝓘, Lists (L_m)_{m∈M})
    F = {}
    t_m = d_m^{ru} ∀ m ∈ M
    while F ≠ O do
        computes a set of processable tasks R.
        if R = ∅ then
            return infeasible
        end if
        for (j,k) ∈ R do
            if (k = 0) then
                S^J(j,k) = max{t_m + d_{j,k}^{se}, a_{j,k}}
            else
                S^J(j,k) = max{t_m + d_{j,k}^{pr}, a_{j,k}, S^J(j,k−1) + d_{j,k−1}^{pr}}
            end if
            t_m = S^J(j,k) + d_{j,k}^{pr}
            R ← R \ {(j,k)}
        end for
    end while
    return compute the corresponding S^M and return (S^J, S^M)
end procedure
```

empty set F, containing all tasks, which are already processed. Also, initially, the next valid processing start of each machine is set to equal the ramping duration. Until the set of finished tasks equals the set of all operations O, the list scheduling tries to schedule the tasks on the assigned machines. To that end, the set R of processable tasks is computed. The set R contains all tasks (j,k), whose predecessors have all already been processed or which have no predecessors. Then, each task in the set of processable tasks is scheduled as early as possible, but after each of its successors. The mapping $\mathcal{S}^J$ describes the corresponding processing starts of the tasks. The algorithm cannot compute a feasible solution if the set R is empty. The corresponding machine states are computed afterward. The respective algorithm is mentioned in Section 2.6.9.

The complex part of this algorithm is the computation of the set of processable tasks. The computation of R requires to consider the execution order, the precedence constraints, and the time windows. Note that only one task per job sequence can be processable per iteration.

Since the energy prices of the job-shop scheduling problem are time-dependent, the optimal schedule can be located within the start, middle, and end or spread within the complete time window. Thus, the list scheduling heuristic, scheduling the tasks as early as possible, could provide rather expensive feasible solutions. To work around the problem of scheduling the tasks as early as possible, we provide a backward list scheduling, which can also detect feasible solutions by scheduling the tasks as late as possible. The backward list scheduling heuristics is similar to the list scheduling heuristic. The difference is that the tasks are processed in reverse order, and the assignment of the processing starts at the end of the time window. List scheduling algorithms benefit from different sets of processable tasks. The sorting of the tasks on the machines mainly influences the processable tasks. Therefore, different sorting comparators may compute different solutions at the same branch-and-bound node. We use the following ordering of the tasks:

- Earliest release date first: Schedule those tasks first which are ready first. The tasks $(j,k),(i,l) \in O^M_{|m}$ on machine $m \in M$ are ordered by

$$a_{j,k} < a_{i,l} \Rightarrow (j,k) \prec_{ERF} (i,l).$$

- Earliest last processing starts first: Prioritize those tasks first that need to be completed earliest. The tasks $(j,k),(i,l) \in O^M_{|m}$ on machine $m \in M$ are ordered by

$$f_{j,k} < f_{i,l} \Rightarrow (j,k) \prec_{EDF} (i,l).$$

Algorithm 8 Backward list scheduling for JSS

procedure BACKWARD LIST SCHEDULING HEURISTIC(Instance $\mathcal{I}$, Lists $(L_m)_{m \in M}$)
  $F = \{\}$
  $t_m = T - d^{rd}_m \ \forall\ m \in M$
  while $F \neq O$ **do**
    computes a set of processable tasks R.
    if $R = \emptyset$ **then**
      return infeasible
    end if
    for $(j, k) \in R$ **do**
      if $(k = O_j - 1)$ **then**
        $\mathcal{S}^J(j, k) = \min\{t_m - d^{pr}_{j,k}, f_{j,k} - 1\}$
      else
        $\mathcal{S}^J(j, k) = \min\{t_m - d^{pr}_{j,k}, f_{j,k} - 1, \mathcal{S}^J(j, k+1) - d^{pr}_{j,k}\}$
      end if
      $t_m = \mathcal{S}^J(j, k) - d^{se}_{j,k}$
      $R \leftarrow R \setminus \{(j, k\}$
    end for
  end while
  return compute the corresponding $\mathcal{S}^M$ and return $(\mathcal{S}^J, \mathcal{S}^M)$
end procedure

- Proposed precedence order of LP-relaxation: try to generate an execution order of the tasks from the fractional solution. The tasks $(j, k), (i, l) \in O^M_{|m}$ on machine $m \in M$ are ordered by

$$\operatorname{argmin}\{t \in [T[_{\mathbb{Z}} \mid \sum_{q=0}^{t} x_{j,k,q} > 0.5\} < \operatorname{argmin}\{t \in [T[_{\mathbb{Z}} \mid \sum_{q=0}^{t} x_{i,l,q} > 0.5\}$$
$$\Rightarrow (j, k) \prec_{LP} (i, l).$$

The list scheduling is effective if the computation of set R is fast. Since the presented orderings are computable with little effort, we use all of the presented orderings to compute different solutions.

4.5.3 Biased Random-Key Genetic Algorithm

Genetic algorithms [GR11, SOMGSOM14, CGT96] and deep-learning approaches [KFH22] have become more and more successful in computing primal solutions to scheduling problems. Since, in the case of the job-shop scheduling problem with flexible energy prices and time windows, the list scheduling heuristics suffer from the fixed execution order of the tasks and possible wrong processing starts, further algorithmic approaches are needed. The idea is to evaluate multiple execution orders and processing starts and take the best one. This idea can be realized by a genetic algorithm. A genetic algorithm is a metaheuristic. These heuristics describe guided search processes exploring the solution space by sampling a subset of primal solutions and keeping the best one.

Within biological evolution, fitter individuals are more likely to pass their genes to further generations than more unfit individuals. The genetic algorithms reflect biological evolution. An individual's fitness is represented by the inverse of the objective value and possible additional penalty terms. Thus, the optimum solution would reproduce its genes as much as possible. An individual is called a **chromosome** within the context of genetic algorithms. A chromosome can be represented by a binary string, which is similar to the encoding of DNA. Another way of representing a chromosome is the usage of a real-valued vector $d \in \mathbb{R}^n$. The set of chromosomes is called a **population**. The chromosomes of a population are mixed and merged over a fixed number of iterations, called **generations**. Two chromosomes, called **parents**, are mixed and merged such that a new chromosome originates. The set of parents is chosen by building up pairs of parents according to the size of the current population. Also, for the selection of the parents, there are multiple rules to follow, for example, the tournament selection, where each parent is the fittest one out of k randomly chosen individuals.

In each generation, the chromosomes of fitter individuals are passed to the next generation. As in the biological counterpart, the chromosomes are passed to the next generation after the functions mutation and crossover are passed.

The function crossover combines two chromosomes and creates a new one. The crossover allows a large degree of freedom. Possible functions are the following ones.

Definition 4.5.1. *Given two chromosomes $A = (a_1, \dots, a_n) \in \mathbb{R}^n$ and $B = (b_1, \dots, b_n) \in \mathbb{R}^n$ with $n \in \mathbb{N}$. The following rules are crossover functions.*

- *Single-point crossover: $A \times B = (a_1, \dots, a_{j-1}, b_j, b_{j+1}, \dots b_n)$ with a randomly chosen $j \in \{1, \dots, n\}$.*
- *Two-point crossover: $A \times B = (a_1, \dots, a_{j-1}, b_j, \dots, b_{k-1}, a_k, a_{k+1}, \dots, a_n)$. with $j < k$ and $j, k \in \{1, \dots, n\}$.*
- *Uniform crossover: $A \times B = (a_1, b_2, a_3, \dots, a_{n-1}, b_n)$.*
- *Interpolation crossover: $A \times B = \sum_{i=1}^{n} \lambda a_i + (1 - \lambda) b_i$ with $\lambda \in [0, 1]$.*

The generation of new chromosomes by crossover can lead to the problem that special traits of individuals cannot be reached if they are not present within the initial population. The mutation step allows the generating of new traits within the algorithm. Within a binary-valued chromosome, the mutation leads to the flip of the trait. One can use bitwise flips in applications where the chromosomes are real-valued.

We are using the implementation of a genetic algorithm of Rodrigo F. Toso and Mauricio C.F. Resende [TR15]. The implementation includes a fixed chromosome encoding, introducing new chromosomes called mutants instead of the mutation operator. The authors introduce the parameters p_e, p_m and p_0 with $p_e + p_m + p_0 = n$, where n is the number of individuals. The fittest p_e elite individuals will also be part of the next generation. There are p_m new mutant individuals part of the population, meaning that p_m mutant individuals will leave the population, and parents will generate p_0 offspring individuals. Introducing mutants and elitism changes the classical genetic algorithm and allows exploring more than the initial combinations of solutions.

In the case of the job-shop scheduling problem with flexible energy prices and time windows, we use the following encoding of the chromosomes.

- The time period encoding: let $C = (c_1, \dots, c_{|O|})$ be a chromosome and $f : O \to \{1, \dots, |O|\}$ a bijective mapping of the tasks to the indices of the chromosome. Then the chromosome C represents the solution as follows:

  $$x_{j,k,t} = 1 \quad \text{for } t = \lfloor a_{j,k} c_{f(j,k)} + (1 - c_{f(j,k)}) f_{j,k} \rfloor \quad \forall\ (j,k) \in O.$$

  The chromosome thus directly describes the processing starts of each task, which also could be invalid. This approach computes a processing start for each task $(j, k) \in O$. There is no consideration of the workload constraints of the machine and the precedence constraints. Therefore, two reasons for infeasibility need to be weighed against each other within the penalty function.
- The time window encoding. Let $C = (c_1, \dots, c_{|O|})$ be a chromosome and $f : O \to \{1, \dots, |O|\}$ a bijective mapping of the tasks to the indices of the chromosome. Then, the time window of each task $(j, k) \in O$ is adjusted as follows:

  $$\hat{a_{j,k}} = \lfloor a_{j,k} c_{f(j,k)} + (1 - c_{f(j,k)}) f_{j,k} \rfloor \quad \forall\ (j,k) \in O.$$

  Using those adapted time windows $[\hat{startsinglejk}, f_{j,k}[_{\mathbb{Z}}$, a list scheduling heuristic using the earliest release date rule tries to compute a feasible schedule. The number of tasks not scheduled within the time window is considered within the penalty function. Thus, the chromosome always satisfies the constraints (3.10c) and (3.10d). After that, a list scheduling heuristic is used to compute a feasible solution. The list scheduling heuristic ensures the precedence constraints and the machines' workload constraints. Therefore, this approach seems more efficient in decoding the chromosomes into feasible solutions. This approach can be seen as a neighborhood search, where one searches for a solution, satisfying the workload and the precedence constraints near the proposed processing starts.

If the chromosomes are decoded into processing starts of the tasks, the objective value of the corresponding solution with ramping and standby can be computed by the shortest path algorithm. Moreover, if the solution of the tasks leads to an invalid schedule, we need to add a penalty term to declare this chromosome unfit.

Let x describe the solution of the list scheduling heuristic with possible non-scheduled tasks. Then, the penalty term of the objective is chosen by

$$\begin{aligned} Penalty = &\sum_{(j,k)\in O}(1 - \sum_{t\in[a_{j,k},f_{j,k}[_{\mathbb{Z}}} x_{j,k,t}) \cdot \max_{t\in[a_{j,k},f_{j,k}[_{\mathbb{Z}}}(\hat{c}_{j,k,t}) \\ &+ \sum_{m\in M}(\sum_{t\in[T[_{\mathbb{Z}}\,:P_t>0} D_m^{st} \cdot P_t + \hat{d}_{m,-d_m^{rd},d_m^{ru}} + \hat{d}_{m,T-d_m^{rd},T+d_m^{ru}}). \end{aligned}$$

The chromosome encoding and evaluation have to consider different infeasibilities. There may be some tasks that are not assigned to a processing start within the list scheduling. To penalize this chromosome, each missing processing start is penalized as much as possible by the term

$$\sum_{(j,k)\in O}(1 - \sum_{t\in[a_{j,k},f_{j,k}[_{\mathbb{Z}}} x_{j,k,t}) \cdot \max_{t\in[a_{j,k},f_{j,k}[_{\mathbb{Z}}}(\hat{c}_{j,k,t}).$$

Since some tasks are not processed, the machine can exploit the fact and use more favorable periods for ramping. Thus, the missing machine state assignments and the corresponding objective costs are penalized by the term

$$\sum_{t\in[T[_{\mathbb{Z}}} D_m^{st} \cdot |P_t| + |\hat{d}_{m,-d_m^{rd},d_m^{ru}}| + |\hat{d}_{m,T-d_m^{rd},T+d_m^{ru}}|).$$

This penalty term is only added to the objective value of the computed solution if a task is not scheduled. Otherwise, the solution is feasible, and the objective describes the fitness of the chromosome. To also handle negative energy prices, the absolute value is considered. Within the implementation of the algorithm and the penalty term, the objective 'always zero' was not considered. The penalty function is not considered to penalize the reason for the infeasibility. Moreover, we decided to penalize the chromosome by the costs of non-scheduled tasks because the combination of the time windows led to this infeasibility.

An additional variant is to consider chromosomes having an entry for each triple (j, k, t) with $(j, k) \in O$ and $t \in [T[_{\mathbb{Z}}$. Then, in the case of real-world applications, the size of the population will be too large to evaluate the objective of each individual since one needs $|O \times [T[_{\mathbb{Z}}\,|$ many chromosomes within the implementation of [TR15].

4.5.4 Dynamic Programming

Dynamic programming is a well-known algorithmic technique mostly designed to solve optimization problems. Like divide and conquer, the dynamic programming approach solves a problem by dividing it into smaller subproblems. The difference between the algorithms is that divide and conquer creates disjoint subproblems, while the dynamic programming approach creates subproblems, which could have sub-subproblems in common with further subproblems. The advantage of this approach is that the information on the solution of the sub-subproblem can be computed and saved. Then, the information is present for further computations.

Dynamic programming applies to problems with the optimal substructure and overlapping subproblems. This property is often called Bellman property [Bel57]. A problem has the optimal substructure if an optimal primal solution can be built from the solutions of its subproblems.

The difference between dynamic programming and enumerating the (exponentially) many potential solutions is that dynamic programming stores the subproblem solutions and need not compute them again. Moreover, the solution of the subproblem is used to compute the solution of the larger problem.

There are different ways to implement a dynamic program:

1. The top-down way: one starts with the original problem and recurses down to subproblems. If the solution of the subproblem is known, then the solution is used to compute the solution of the larger problem. If the subproblem is not solved yet, the subproblem is solved, and the result is stored for future use.
2. The bottom-up approach: The algorithm starts by solving the subproblems. It uses the solutions of the subproblems to build solutions of subproblems of a higher level until the original problem is solved.

The solution process creates a solution for the global problem by using the information of the solution of a subproblem in a recursive way. The main characteristics of a dynamic programming approach are described as the following three characteristics:

1. *Stages:* A dynamic program is structured into multiple stages. The stages are solved sequentially, and each stage is an optimization problem itself.
2. *States:* The states reflect the information required to assess the current decision's consequences upon future actions fully. The state conveys enough information to make future decisions without regard to how the problem reached the current state. The choice of the states is a decision in algorithm design, and a good choice is crucial since the number of concerned subproblems can become huge and thus also the number of different state variables.
3. *Recursive optimization:* The usage of a recursive optimization procedure, which builds to a solution of the complete problem by first solving the problems of the single stages and sequentially collecting the information from one stage at a time and solving an additional one-stage problem until the overall optimum has been found.

To describe the dynamic programming approach formally, we introduce the *return* of a stage n by $f_n(d_n, s_n)$. The parameter $d \in D_n(s_n)$ describes a permissible decision out of the set of all valid decisions in state s_n. The index n describes the remaining stages out of the finite maximum number of stages $N \in \mathbb{N}$. The *transition function* returns the next state such that, given s_n, the state of the process with n stages to go, the subsequent state of the process with $(n-1)$ stages to go is given by

$$s_{n-1} = t(d_n, s_n).$$

Job-Shop Scheduling by Dynamic Programming

We provide a dynamic programming approach to the job-shop scheduling problem with flexible energy prices and time windows. Therefore, we assume a fixed execution order of the tasks to reduce the number of solutions and sub-solutions by the dynamic program. Since the execution order of the tasks is not fixed in general, we have to derive a total order of the tasks by ourselves. Note that the derived total order mainly influences the outcome and the resulting objective value.

Definition 4.5.2 (Total order). *The relation $\prec^{TO}$ describes a total order of all pairs of distinct tasks $(j,k),(i,l) \in O$, if the digraph $D^{TO} = (V^{TO}, A^{TO})$ with*

$$\begin{aligned} V^{TO} :=& \{(j,k) | (j,k) \in O\} \\ A^{TO} :=& \{((j,k),(i,l) \mid (j,k),(i,l) \in O, (j,k) \neq (i,l),\ (j,k) \prec^{TO} (i,l)\} \end{aligned}$$

is an acyclic digraph, whose underlying undirected graph $G = (V^{TO}, E^{TO})$ is connected.

The usage of a total order shrinks the number of possible solutions and possibly prevents us from finding the optimal solution caused by a misleading execution order.

The definition of a total order $\prec^{TO}$ on the set of tasks O lead to a chain of tasks $(j_0, k_0) \prec^{TO} \cdots \prec^{TO} \rightarrow (j_{n-1}, k_{n-1})$, where $(j_i, k_i) \prec^{TO} (j_c, k_c)$ for all $i < c$ holds. Now, we declare each of the classical dynamic programming parameters by the usage of our job-shop scheduling notation. We assume, that $|O| = n$ holds

- The parameter s_i is a $i+1$-dimensional vector describing the starting periods of all tasks (j_c, k_c), $0 \leq c \leq i$.
- The parameter $D_j(s_i) \subseteq [a_{j_i,k_i}, f_{j_i,k_i}[_{\mathbb{Z}}$ is a subset of the valid processing starts of task (j_j, k_j). The set $D_j(s_i)$ does not equal the set of possible processing starts of task (j_j, k_j) since some processing starts are possibly no longer allowed after some iterations of the dynamic program.
- The parameter d_i describes the processing start of task (j_i, k_i) , $0 \leq i < n-1$.
- The parameter $cost_i$ describes the generated costs by starting the processing of the tasks (j_c, k_c) $0 \leq c \leq i$ within the periods in s_i. The additional machine costs between the tasks and the initial phase of the machine are also considered.
- The parameter $R_m^{d(i}$ describes the last occupied period of machine m after scheduling task (j_i, k_i) in period d_i.
- The mapping $I : O \rightarrow \mathbb{N} \cup \{0, -1\}$ returns the index of the predecessor of a task, if any exists. Otherwise, the function returns the value -1.
- The mapping $M(i)$ returns the associated machine of task (j_i, k_i) for $0 \leq i \leq n-1$.
- The parameter
$$best_i : M \times [T[_{\mathbb{Z}}^i \times [T[_{\mathbb{Z}} \rightarrow \mathbb{R}$$
describes the local optimal costs for starting the processing of task (j_i, k_i) in Period $d(i)$. The computation of an initial ramping costs is included in the computation of $best_i$, if the task (j_c, k_c) is the first one on machine $M(i)$.

- The parameter $best\_final(M(i), s_i))$ describes the best objective costs for realizing the final ramp-down if the tasks are fixed to start processing in the period s_i.
-

Using these parameters and auxiliary functions, we get the following procedure:

$$f_i(d_i, s_i) := \begin{cases} \min\limits_{d_{i-1} \in D(s_{i-1})} \Big(f_{i-1}(d_{i-1}, s_{i-1}) + best_i(M(i), d_{i-1}, d_i) \Big) + \hat{c}_{j_i,k_i,d(i)} & i \geq 1 \\ best_i(M(i), 0, d_i) + \hat{c}_{j_i,k_i,d(i)} & sonst \end{cases}$$

The function $f_i(d_i, s_i)$ is the recursive function. The function describes that the best objective when scheduling task (j_i, k_i) can be computed from the best objective when scheduling task (j_{i-1}, k_{i-1}). The assignment of a task (j_i, k_i) to a period d_i in the stages $n-i$. The states of stage i are described by $d(i)$, denoting processing starts of task (j_i, k_i).

To reduce the number of subproblems within the computation of

$$\min \left\{ f(d_n, s_n) + \sum_{m \in M} best\_final(m, s_n) \middle| d_n \in D(s_{n-1}) \right\}.$$

We can define a so-called *dominance criterion.*

Theorem 4.5.3 (Dominance criterion)**.** *Let L be a total ordered list of all tasks O and l and q two states of the same stage i, with objectives $cost_q$ and $cost_l$. The state l need not to be observed within the algorithm if*

$$R^q_m \leq R^l_m \quad \forall\ m \in M$$
$$cost_l \geq cost_q + best(m, R^q_m, on, R^l_m, on) \text{ with } m = m_{j,k} \text{ and } (j,k) = (j_i, k_i).$$

Proof. Let q and l be to states of the same stage i with

$$R^q_m \leq R^l_m \quad \forall\ m \in M$$
$$cost_l \geq cost_q + best(m, R^q_m, on, R^l_m, on) \text{ with } m = m_{j,k} \text{ and } (j,k) = (j_i, k_i).$$

The solution provided by state q needs fewer periods on each machine to process the same subset of tasks. Therefore, each possible configuration that is feasible for state l is also feasible for state q. Moreover, by adding standby and breaks, the parameters R^q_m and R^l_m can be aligned for each $m \in M$ and the alignment of the used periods on the machines results in a higher objective of state l. Thus, aligning state q to state l leads to a cheaper solution for processing the same tasks. Thus, the state l can be discarded since we can recreate the same solution more cheaply. □

Speed-up Techniques

The presented algorithms create $\mathcal{O}(T^{|O|})$ many (sub)solutions, neglecting the discarding of subproblems. Different techniques are necessary to reduce the number of possible solutions and, thus, the number of necessary computations. Some approaches are briefly discussed in the following part.

In the context of the job-shop scheduling problem with flexible energy prices and time windows, the time windows of the tasks can be discretized more coarsely. Instead of using the allowed processing starts $t \in [a_{j,k}, f_{j,k}[_{\mathbb{Z}}$ for each task $(j,k) \in O$, we could use the coarsely time window $\{t \mid t = a_{j,k} + l \cdot \Delta \mid t \in [a_{j,k}, f_{j,k}[_{\mathbb{Z}}\ \}$ and $\Delta \in \mathbb{N}$. The optimal solution could also be missed since we shrink the number of possible solutions.

A second speed-up technique uses an upper bound and an approximation for the objective value to decide whether a good or improving solution can still be found. Consider the state l of stage q. Then, the resulting objective of state l can be approximated by

$$cost_l + approx(l)$$

where $approx(l)$ provides a lower bound to the optimal objective value of scheduling the remaining l tasks and ramping the machine down. A possible lower bound can be:

$$\begin{aligned} approx(l) = \sum_{m \in M} & \min\{\hat{d}_{m,t_0,t_1} \mid (t_0, t_1) \in B_m,\ t_1 = T + d^{ru}_m\} + \\ & \sum_{v=l}^{|O|} \min\{\hat{c}_{j,k,t} \mid t \in [a_{j,k}, f_{j,k}[_{\mathbb{Z}}\ , (j,k) = I(v)\}. \end{aligned}$$

The value $approx(l)$ can be computed initially. The upper bound could be the objective of the incumbent primal solution or the current fractional solution plus 10 %.

4.5.5 Local Search Algorithms

The idea of local search is intuitive: the algorithm starts with a feasible solution, and it is assumed that further solutions will be found by making small changes to this solution. If there is an improving solution, this process will be reiterated with a new incumbent primal solution until no more improvements are found. The idea of local search has been already successfully applied in the field of scheduling, see, for example, [DT93, NS05, Yin04].

To describe a local search algorithm formally, some notions are required. The set of feasible primal solutions is denoted by $\mathcal{S}$. The objective function $f : \mathcal{S} \to \mathbb{R}$ maps the solutions to their objective values. The mapping $N : \mathcal{S} \to 2^{\mathcal{S}}$, $x \mapsto N(x)$ defines the neighborhoods of the solutions $x \in \mathcal{S}$. The solution $x \in \mathcal{S}$ is called a local minimum concerning the neighborhood $N(x)$ if $f(x) \leq f(y)$ for all neighbors $y \in N(x)$. The algorithm tries to iteratively improve the current solution by exploring the neighborhood to find improving solutions.

The quality of the solution of a local search algorithm depends on the initial solution and the neighborhood. A large neighborhood offers improving solutions with a higher possibility than small neighborhoods. However, exploring a large neighborhood may be more time-consuming than exploring a small neighborhood.

There are so-called threshold algorithms, where a neighbor is also accepted as the new incumbent if the objective gain (or loss) is below a certain threshold.

In our case, the neighborhood function and the solutions can be represented in different ways. We are given a schedule $\mathcal{S}^J : O \to [T[_{\mathbb{Z}}$. For $(j,k) \in O$, the mapping $js(j,k)$ describes the successor of (j,k), if it exists, in the schedule on the machine $m_{j,k}$. The parameter $jp(j,k)$ describes the direct predecessor of task (j,k) if it exists within the schedule S on the machine $m_{j,k}$. Two tasks $(j,k),(i,l) \in O^M_{|m}$ are adjacent, if there is no task $(i_3,l_3) \in O^M_{|m}$, with $S_{j,k} < S_{i_3,l_3} < S_{i,l}$ or $S_{i,l} < S_{i_3,l_3} < S_{j,k}$. The execution order

$$\Omega_S := \Big\{\big((j,k),(i,l)\big) \in O \times O \mid (j,k) \text{ and } (i,l) \text{ are adjacent}\Big\}$$

is defined as the set of pairs of adjacent tasks. The neighborhoods are defined on the execution order Ω_S of a schedule S.

A simple definition of the neighborhood of S can be defined by

$$N^{\Omega}(S) = \Big\{T \in \mathcal{S} \Big| |\Omega_S \cap \Omega_T| \leq 6\Big\}.$$

The value is 6 since there are three changes from deleting the old order of two tasks and three changes to introduce the new order of the two tasks on the machine. All other pairs are still valid. This neighborhood considers the execution order of the tasks on the machines, and we call two schedules neighbors if the execution order of schedule $\mathcal{S}^J$ can be transformed into the execution order of schedule T by interchanging the execution order of at most one pair of tasks.

Another version of a neighborhood of $\mathcal{S}^J$ can be done by encoding the schedule S by the corresponding solution of the time-indexed solution and set.

$$N^x(S) = \{T \in \mathcal{S} \mid \sum_{(j,k)\in O} \| \sum_{t \in [T[_{\mathbb{Z}}} x^S_{j,k,t} - x^T_{j,k,t} \| \leq 2\}.$$

This neighborhood is allowed to change the processing start of one single task. There is a more complex neighborhood, which is discussed within the overview article [VAL94] by R.J.M. Vaessens and E.H.L. Aarts, J.K. Lenstra. In the following, we only consider our implemented functions. The neighborhood N^{Ω} is explored by choosing a neighboring orientation. This orientation describes the execution order of the tasks and the neighboring orientation differs in only two positions from the currently best orientation. Then, we compute a schedule using the orientation of the tasks, for example, by list scheduling or a dynamic program. The schedule, which is computed, is only one solution that can be generated from the fixed processing order. The computed schedule need not be the best one.

Then, neighborhood N^{Ω} only contains the computable solutions of the neighborhood. The complete enumeration of the neighborhood would be too large.

Thus, we consider only representatives for the solutions to an orientation. The cost function computes the objective value of a solution $f : \mathcal{S} \to \mathbb{R}$, and we take the risk that we have chosen a representative with a (too) large objective. The local search algorithm with threshold is described below. One can consider restarting the complete Algorithm 9 when finding a new solution.

Algorithm 9 General threshold accepting local search

procedure GENERAL LOCAL SEARCH(Instance $\mathcal{I}$, initial solution S)
  repeat
    Derive local information about variable bounds and start time windows.
    for representative $S' \in N(S)$ **do**
      if $f(S') - f(S) < threshold$ **then**
        $S \leftarrow S'$
      end if
    end for
  until maximum number of iterations is reached
end procedure

The variable bounds' local information must be computed initially to consider the tasks' local time windows. This is because the local time windows of the tasks allow the computation of different solutions at each branch-and-bound node by list scheduling. This is necessary since list scheduling needs local information about the time windows to change the processing starts of the tasks.

Also, if the dynamic program is used as a heuristic, only a subset of the processing starts is allowed. Thus, the dynamic program will also profit from the computation of the tighter time window.

However, then the solution can only become a locally optimal solution.

Local Search by Dynamic Programming

Local search algorithms are useful tools in integer programming. However, the exploration of the neighborhood's solutions may be inefficient. The resulting subproblems are often also MILPs and only slightly easier to solve. Suppose we are given a primal solution encoded by processing starts for each task. We can check whether the solution could be improved by shifting the processing start of a subset of tasks to the left or the right. Consider the task $(j,k) \in O$. Then, all tasks $(i,l) \in O \setminus \{(j,k)\}$ are assumed to be fixed to their period of the incumbent solution. The best position of (j,k) in relation to the fixed task $(i,l) \in O \setminus \{(j,k)\}$ is computed by dynamic programming. Therefore, for a given primal solution $\mathcal{S}^J$, the idea is to allow an shift $\delta \in [T[_{\mathbb{Z}}$, such that we search for an improving solution in $[sol_{j,k} - \delta, sol_{j,k} + \delta[_{\mathbb{Z}}$ for each $(j,k) \in O$.

The dynamic program is efficient since only one task needs to be considered within the computation. All further tasks are fixed. Thus, because of the domination criteria, one need not compute the objective of the schedules from scratch. Thus, this approach is more efficient.

Algorithm 10 describes the local search algorithm with dynamic programming in pseudo-code.

Algorithm 10 General threshold accepting local search

procedure LS BY DP(solution $\mathcal{S}^J$, neighborhood size δ)
  for $(j,k) \in O$ **do**
    set $[a_{j,k}, f_{j,k}[_{\mathbb{Z}} = [S_{j,k} - \delta, S_{j,k} + \delta[_{\mathbb{Z}}$
    set $[a_{i,l}, f_{i,l}[_{\mathbb{Z}} = [S_{i,l}, S_{i,l} + 1[_{\mathbb{Z}}$ for each $(i,l) \in O \setminus \{(j,k)\}$.
    Compute the best solution by DP using the derived time windows
    if new incumbent **then**
      Update incumbent solution
    end if
  end for
end procedure

The neighborhood search algorithm by dynamic programming is efficient since only one task is adjustable. The computation complexity of this neighborhood search algorithm is $\mathcal{O}(T)$, and the algorithm can be extended to multiple tasks. However, this approach can only succeed if the incumbent is not locally optimal for the fixed execution order.

4.5.6 Diving Heuristics

Diving heuristics can be seen as methods that extend a sub-solution by iteratively diving into one single direction of the branch-and-bound tree. The diving heuristic is summarized as the depth-first search walks through the branch-and-bound tree. In contrast, a predefined variable fixing rule generates the branch-and-bound tree until a feasible primal solution can be computed or the resulting problem is infeasible. After each fixation, the resulting relaxation will be solved again.

There are many diving heuristics. One can imagine that each possible branching rule can be applied as a depth-first search heuristic algorithm. However, these heuristics do not consider and recognize the problem-specific aspects. The analysis of the problem's complexity has shown that the computation of the processing starts is a hard problem. In contrast, the computation of the corresponding machine states is possible in polynomial time. Thus, the diving heuristic must focus on scheduling the tasks. We devised and implemented a diving heuristic, which fixes the task variables to generate a near-optimal solution. Our diving algorithm aims to fix the task variables and solve the relaxation after each fixation. The Algorithm 11 shows the pseudocode of our implemented diving heuristic: The algorithm iteratively fixes the task variable, with the largest fractional value

Algorithm 11 Iterative rounding on task variables

```
procedure ITERATIVE ROUNDING(fractional solution x)
    repeat
        compute (j*, k*, t*) = argmax{x_{j,k,t} | (j,k) ∈ O, t ∈ [T[_Z , x_{j,k,t} ∉ {0,1}}.
        fix x_{j*,k*,t*} = 1
        resolve resulting relaxation P
        if relaxation is infeasible then
            return
        else
            set x = x'
        end if
    until x is integral
    compute S^J and the corresponding best S^M. return (S^J, S^M).
end procedure
```

until the resulting problem is infeasible or a primal solution is computed.

Algorithm 12 Iterative rounding on task variables

```
procedure ITERATIVE ROUNDING(fractional solution x)
    repeat
        compute (j,k) = argmin{r(i,l) − l(i,l) | (i,l) ∈ O, r(i,l) − l(i,l) > 0}
        compute t = ⌊∑_{q∈[a_{j,k}, f_{j,k}[_Z} x_{j,k,q} · t⌋
        fix x_{j,k,t} = 1
        resolve resulting relaxation P
        if relaxation is infeasible then
            return
        else
            set x = x'
        end if
    until x is integral
    compute S^J and the corresponding best S^M. return (S^J, S^M).
end procedure
```

The second variant fixes the task (j, k), for which the processing start is nearly fixed. Both diving heuristics focus on scheduling the tasks. The best corresponding objective value is computed automatically.

There was a need to implement these diving algorithms since classical diving heuristics do not distinguish task and break variables. Thus, classical diving heuristics could create dives by fixing standby or break variables. Those algorithms would be misleading since the integral solution is defined by the processing starts of the tasks, see Section 3.2.7. Our implementation knows this property and only considers the task variables in diving.

Summarizing Comment

The implementation contains different list scheduling heuristics, which take the fractional solution and try to construct an integer solution from the information of the LP-relaxation. Multiple approaches were presented to explore neighborhoods of incumbent solutions to improve the solutions. These approaches are useful to provide a primal solution without solving an ILP. The biased random key genetic algorithm is a metaheuristic that describes a guided search process within the space of all schedules. A more expensive algorithm is the dynamic programming approach requiring a total order of the tasks. But then, an optimal solution regarding the fixed execution order can be computed. In combination with the presented diving heuristics, multiple aspects are considered within the solution process: deriving solutions from the execution order of the tasks and computing solutions by iterative rounding based on the current fractional values. The variety of implemented heuristics offers a large potential to compute near-optimal solutions fast. Combined with the implemented MILP heuristic, the solution process can compute near-optimal solutions early.

Chapter 5

Implementation and Computational Experiments

This section introduces the details of the implemented algorithms and the different effects of our algorithms on solving statistics.

5.1 Implementation

The algorithms, presented in Section 4.1, 4.2, 4.3, 4.4 and 4.5 are implemented in C++ using the interfaces of SCIP 8.0.3 [BBC+22]. The LP-solver of SCIP is GUROBI [GO22]. The algorithms use the implemented functions of SCIP and further straightforward implementations without unconventional or parallel approaches and techniques. The devised algorithms are presented by their pseudocode within this thesis. Within the heuristics, the framework of Resende [GR11] is used to implement a genetic algorithm. However, the algorithm is realized without unconventional implementations.

The problem formulation is generated using the problem data interface of SCIP. The problem formulation includes the knapsack constraints (3.19), (3.20), (3.20) and (4.2). The precedence constraints (3.10c) are implemented using constraint handler. The constraint handler separates violated precedence constraints and detects locally valid precedence relations. The presolving and propagation rules are implemented using the propagator interface of SCIP. All mentioned presolving and propagation rules are implemented as described within this thesis. The interfaces for branching rules are used to implement the assignment constraint branching (4.43) and the branching on machine activity (4.40) and (4.39). The conflicts (4.44) are added to the conflict graph during presolving. The GUB cover constraints (4.53) and (4.55) are separated using the separator interface of SCIP. The possible inequalities are efficiently enumerated and the violated ones are added to the cut pool of SCIP. The column generation approach is realized by using the pricer interface of SCIP. The efficient enumeration scheme 4.4.1 of the break variables is implemented. Each of the mentioned heuristics in 4.5 is implemented using the heuristic interface of SCIP. The frequency and the priority of the heuristics is documented in Appendix A.1.

5.2 Parameter Choices

The default SCIP parameters are changed so that the default heuristics are deactivated except for the **simplerounding** heuristic. Experiments have shown that those heuristics are not as efficient as this heuristic. The propagation of **vbounds** and **probing** is forbidden since those techniques either do not increase the efficiency of the solving process or are too time-consuming. Within the presolving stage, the default presolving settings are used, and SCIP decides automatically when the presolving is aborted. When separating known valid inequalities, most pre-implemented separators are deactivated since they are too time-consuming and less efficient. In that end, the frequency of calling the **clique-separator** and the **implied bound separator** is set to 1 since these separators strongly influence the solving process. However, separating valid inequalities is only triggered at nodes with **maxbounddist=0**. The branching algorithm and the corresponding branching rules are the implemented branching rules. The preferred branching rule is workload branching. We compute the fractional interval by the highest fractional workload criteria (W). A weighted mean of the fractional workload classically determines the branching period. The second branching rule is the branching on assignment constraint. By default, the task to branch on is selected by computing the product of activity and fractional spread. A weighted mean determines the period. There will be no need to implement a third branching rule. The precedence constraints (4.2) are used to implement the precedence relation of the tasks within the model. Moreover, they are used to check the validity of computed solutions. To strengthen the LP-relaxation, the classical precedence constraints (3.10c) are implemented to be **lazy-constraints**. Therefore, the huge number of constraints does not affect the solving process in the first place. Moreover, the detection and the separation of valid precedence constraints (3.10c) is triggered at each node as much as necessary. Moreover, the search for valid precedence constraints is triggered if no initial precedence constraints are violated. The separation of the GUB covers constraints is also triggered at each node with **maxbounddist=0**, and only a subset of the computed violated constraints is filtered and added by SCIP to the problem. The implemented heuristics are not called at each node within the branch-and-bound tree. The frequency of calling our diving heuristics is set to **freq** $= \mathbf{4}$. However, the heuristic must be called if the local optimality gap is small (e.g., $1e-2$) and the branch and multiple branches are already pruned. The computation of the number of pruned branches is determined by **the number of leaves divided by the number of nodes** < 0.8. The node selection of the branch-and-bound algorithm is set to **best first search** to strengthen the dual bound and not to discover non-optimal branches in the first place. The settings of the column generation algorithm require to be completely different since most propagation and presolving rules, as well as cutting planes are not valid anymore since they need to be considered within the pricing algorithm. Thus, in the case of column generation, all cutting planes by SCIP as well as the presolving by SCIP are disabled. In addition, the dual reductions are forbidden. In the experimental results, the attached default settings Appendix A.1 are always used, and it is mentioned in the captions whether a setting has been changed, for example, the branching rule.

5.3 Generation of Test Instances

The test instances are based on the benchmarks of Lawrence [Law84]. The job sequences and the task-to-machine assignment are copied. The instances of Lawrence only include processing times. Let $p \in \mathbb{N}$ be the processing duration of task (j,k) of an instance of Lawrence. Lawrence's processing times are divided into setup and processing times in a ratio of 1 to 2. Since large time windows impact the solution times, we divide the resulting processing times by 10 and rounding up the resulting value. Then, the setup and the processing duration are computed by

$$d^{se}_{j,k} = \frac{1}{30}p \text{ and } d^{pr}_{j,k} = \frac{2}{30}p.$$

The ramping durations are set to equal the mean of the processing and setup durations on the corresponding machine. The energy demand of the different machine states is randomized as follows:

$$D^{ru}_m = \text{rand}(1,20) \tag{5.1}$$
$$D^{rd}_m = \text{rand}(1,20) \tag{5.2}$$
$$D^{off}_m = 0 \tag{5.3}$$
$$D^{pr}_m = \text{rand}(5,20) \tag{5.4}$$
$$D^{se}_m = \text{rand}(2, \frac{D^{pr}_m}{2} + 1) \tag{5.5}$$
$$D^{st}_m = \text{rand}(1, \max(D^{pr}_m, 2)). \tag{5.6}$$

The operator $\text{rand}(x,y)$ returns a random integer in the set $\{x,\dots,y\}$. The numbers are uniformly distributed.

The objectives of the instances are different realistic objectives of different periods derived from the website of Bundesnetzagentur [Bun21]. The objectives are labeled as follows:

- The shortcut _0_ denotes the disturbed and scaled sin curve, computed by $C_t = \lfloor \sin(\pi \cdot t/T) + 1) \cdot 10 \rfloor$, for all $t \in [T]$.
- The shortcut _1_ denotes constant energy prices and results in the objective of minimizing the consumed energy.

- The shortcut _7_ denotes the real energy price from March 1st to May 31st within the year 2021.
- The shortcut _8_ denotes the energy price of Germany from October 1st to October 13th, 2021.

Remark 5.3.1. *The objectives were simply numbered. There are other objectives with the missing indices, but these objectives are only for test purposes and are of no further relevance. The index of the target function has no further meaning and is only used for differentiation.*

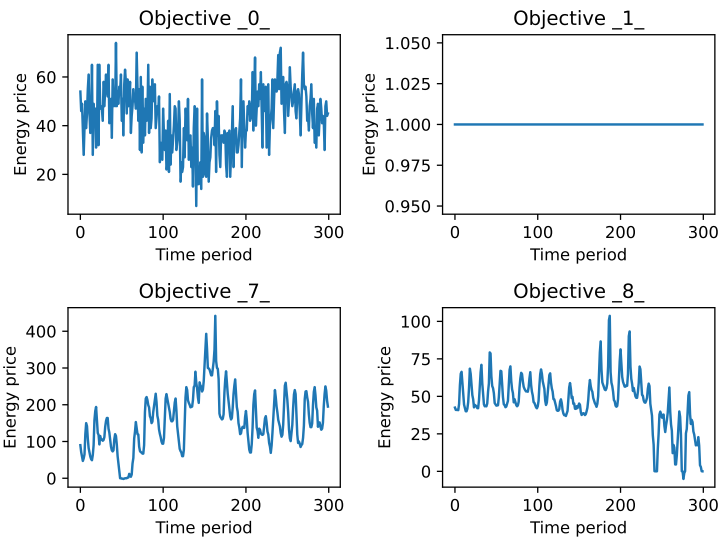


Figure 5.1: Visualization of the different energy costs

The time window T is set to $\sum_{(j,k)\in O}(d^{pr}_{j,k}+d^{se}_{j,k})\cdot\frac{1}{n_M}\cdot\gamma$ and γ is chosen in $\{1.5, 1.75, 2, 2.25\}$ for the shortcut s,m,l,h. Moreover, the instances are generated so that the energy demand is scaled by a second factor $\gamma_2 \in \{0.5, 1, 1.5, random\}$. If γ_2 equals random, then the energy demand is multiplied by a random number in $[1, 2]$. The instance $laXX$ of Lawrence is transformed into the instance

$$laXX\_objective\_energyDemandScale\_TimewindowScale.$$

The instances are built to consider different time window lengths for the same instance. Thus, the complexity of the problem is increased by the time window size. The different lengths of the ramping durations lead to less use of breaks. The increase in the objective function also leads to a higher difference between the costs of breaks and standby costs. Thus, higher energy demand for breaks leads to less use of breaks within integral solutions. The chosen constants are arbitrarily chosen without any reference.

Note that each mentioned technique is implemented in *C++*. Thus, no explicit overview of the implemented techniques is provided.

5.4 Experimental Results

Within this section, we discuss the experimental results of the implemented algorithms. We present different results, visualized by figures, and comment and discuss the arising questions. In the beginning, the problem sizes of the break-based and the stated-based formulation are considered. Additionally, the quality of the break-based formulation is discussed. Then, the statistics of Gurobi solving our break-based formulation are analyzed. After that, the different components of the implemented branch-and-bound algorithm are analyzed to determine whether they are crucial to solving the problems efficiently. At the end of this chapter, there is a comparison of Gurobi's performance and the performance of our implementation to show that the implemented algorithms lead to a considerable solution algorithm, which is comparable to the solution time of parallel programmed commercial solvers.

5.4.1 Comparison of the State-Based and the Break-Based Formulation

The problem formulation (3.1a)–(3.1k) and the partial break-based model (3.10a)–(3.10h) use different variable sets and also different constraint sets. Thus, the problem sizes may differ. The Table 5.1 shows the number of variables and constraints of the break-based formulation (3.10a)–(3.10h) and the state-based formulation (3.1a)–(3.1k).

Table 5.1: Average number of variables and constraints of different formulations.

instance	break based model		state based model	
	variables	constraints	variables	constraints
laXX_s_s	15079	7380	8282	8379
laXX_s_m	24358	9039	9546	9722
laXX_s_l	33847	10271	10874	11133
laXX_s_h	47296	12441	12138	12476
laXX_m_s	12111	6715	8282	11788
laXX_m_m	21390	8374	9546	13684
laXX_m_l	32474	10117	10874	15676
laXX_m_h	44328	11776	12138	17572
laXX_l_s	9372	3807	8519	9002
laXX_l_m	18645	7802	9782	17113
laXX_l_l	29713	9543	11108	19600
laXX_l_h	41557	11201	12372	21968
laXX_r_s	12166	5966	8636	10481
laXX_r_m	21400	8283	9846	13507
laXX_r_l	32131	9885	11231	15996
laXX_r_h	44418	11679	12414	17298

Table 5.1 shows the averaged model size of our break-based model and the state-based formulation for the different settings. The average value is computed for each problem size. The table shows that the break-based model significantly uses more variables. The number of variables of the state-based formulation linearly depends on the time window, while the break-based model has a quadratic dependency. The number of constraints of both formulations increases with the linear dependency of the time window size. This table clearly shows the number of constraints growing faster than the number of variables in the case of the state-based formulation. In the case of the break-based model, the number of constraints is growing at the same rate. Thus, the break-based model requires more variables and fewer constraints than the state-based model initially.

We presented various presolving rules, which reduce the number of break variables. The resulting problem sizes are presented in Table 5.2. We use the mapping R to describe the expression

$$R(solver, instance) = \frac{\text{remaining variables after presolving by solver}}{\text{Number of initial variables}}$$

Table 5.2: Average number of variables and constraints of different presolved formulations.

instance	break-based model			state-based formulation	
	#Variables	R(Gurobi,:)	R(Scip+,:)	# Variables	R(Gurobi,:)
laXX_s_s	15079	0.818	0.407	8282	0.789
laXX_s_m	24358	0.818	0.437	9546	0.866
laXX_s_l	35442	0.845	0.464	10874	0.929
laXX_s_h	47296	0.884	0.484	12138	0.975
laXX_m_s	12111	0.73	0.309	8282	0.706
laXX_m_m	21390	0.716	0.261	9546	0.793
laXX_m_l	32474	0.727	0.264	10874	0.865
laXX_m_h	44328	0.773	0.285	12138	0.918
laXX_l_s	9372	0.449	0.345	8519	0.391
laXX_l_m	18645	0.635	0.231	9782	0.713
laXX_l_l	29713	0.642	0.196	11108	0.792
laXX_l_h	41557	0.67	0.184	12372	0.851
laXX_r_s	12166	0.697	0.355	8636	0.608
laXX_r_m	21400	0.739	0.308	9846	0.761
laXX_r_l	32131	0.736	0.3	11231	0.817
laXX_r_h	44418	0.777	0.328	12414	0.891
means	27618	0.729	0.322	10349	0.792

Table 5.2 shows the problem sizes after our and Gurobi's presolving. The number of initial variables and the relative number of remaining variables after Gurobi's default

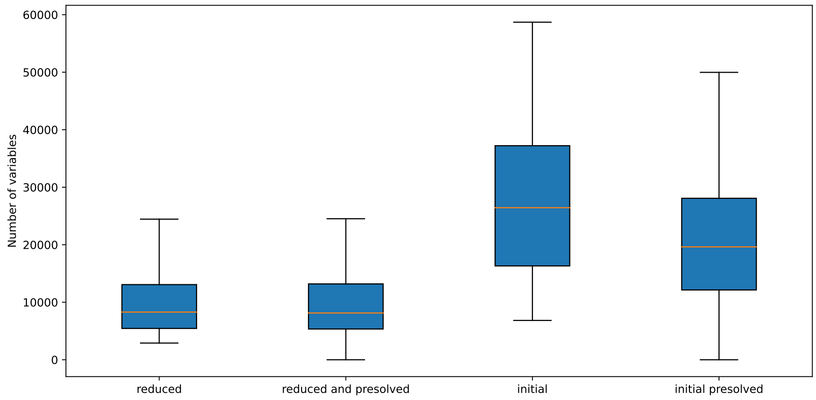


Figure 5.2: Comparison of the number of variables of the break-based model in a box plot. The orange line shows the median, the blue box describes the first and third quartiles, and the black lines denote the outliers. The considered formulations are the formulation, which is initially reduced by our presolving, the initially reduced formulation, presolved by Gurobi, the Dantzig–Wolfe reformulation with all variables and the initial break-based model presolved by Gurobi.

presolving and the relative number of remaining variables after our presolving. Analogously, the initial number of variables and the relative number of remaining variables of the state-based formulation are presented. A presolving algorithm is stated to be more efficient if the relative number of remaining variables is small. One can see that initially, the break-based formulation has more variables (factor $2 - 3$) than the state-based formulation. The default presolving efficiency of Gurobi can reduce the number of variables by 25.5%. In contrast, the presented variable reduction in Section 4.1 allows a problem size reduction of 67.8%. After usage of our presolving, the break-based formulation has a similar size as the state-based formulation presolved by Gurobi. Thus, the disadvantage of the problem size can be compensated. We do not discuss the impact of presolving rules for constraints since the formulation (3.10a)–(3.10h) only includes necessary constraints, except the precedence constraints, which are considered lazy cuts. Thus, the number of constraints cannot be reduced significantly anymore. Another research question is whether presolving by Gurobi can further reduce the problem we have already presolved. If this is not the case, then we could describe all significant reductions with combinatorial conditions. Figure 5.2 visualizes the effect of the aggressive presolving by Gurobi on our presolved formulation. Figure 5.2 shows the distribution of the number of variables of different stages of the presolving process. There is the number of initial variables (initial), the number of variables after our presolving (reduced), the number of variables after our presolved formulation is presolved again by Gurobi (reduced and presolved), and last but not least, the number of variables if Gurobi presolves the initial model. One can see that our presolving can be strengthened by Gurobi's additional presolving. However, the impact is not significant. In addition, one can see that our presolving outperforms the default presolving of Gurobi. Most times, Gurobi is able to detect by presolving whether the problem is infeasible or not. Then, the presolving is able to reduce the complete set of variables. Figure 5.2 shows that these are, among others, the cases where Gurobi can reduce the number of variables of the presolved formulation. Since our presolving is not considered to detect the infeasibility of the instance, the number of variables will also not be reduced until no variables exist anymore. Since our presolving is able to eliminate 66% of the variables, it is interesting which rules are successful. The effect of the different presolving rules at the different problem sizes is shown in Table 5.3.

Table 5.3: Reduction of the break-variables by the different presolving rules.

instance	#breaks	length	non-usable	unnecessary	bin packing	objective
laXX_X_s_s	24092	12880	1200	1052	4	1748
laXX_X_s_m	32496	12880	1912	1688	4	1748
laXX_X_s_l	42672	12880	2656	2360	4	1748
laXX_X_s_h	53664	12880	3364	3000	4	1748
laXX_X_m_s	24092	15500	2252	768	16	3404
laXX_X_m_m	32496	15500	4076	1404	16	3736
laXX_X_m_l	42672	15500	5980	2080	16	3736
laXX_X_m_h	53664	15500	7792	2724	16	3736
laXX_X_l_s	22879	17035	2261	505	44	3097
laXX_X_l_m	30857	17035	4885	1117	44	4797
laXX_X_l_l	40510	17035	7640	1758	44	4977
laXX_X_l_h	50944	17035	10272	2369	44	4977
laXX_X_r_s	24092	15381	1944	769	25	2807
laXX_X_r_m	32496	15461	3564	1388	22	3408
laXX_X_r_l	42672	15742	5821	2023	27	3817
laXX_X_r_h	53664	15373	7339	2641	24	3496

Table 5.3 shows the number of variable reductions broken down according to the presolving rules. All presolving rules are only applied if the break length does not exceed the trivial bound of the knapsack constraint (3.18). Then, each variable has to pass each presolving rule to get the efficiency of the presolving unrelated from their order. This evaluation shows that non-usable breaks are a large part of the redundant variables. Non-usable breaks can be detected by probing. Fixation of the break to one leads directly to an infeasible problem. The elimination of unnecessary breaks, which require positive energy prices, only reduces the small subset of variables. However, these variables will never used in optimal solutions. The bin-packing and the small time window reduction perform badly as a presolving rule due to the fact that these rules are only high-performing if the time windows are small. Thus, those rules are considered to be propagation methods within the branch-and-bound tree. Since the branch-and-bound tree will also include assignment constraint branchings (4.43), some time windows will be reduced, and thus, these propagation methods will detect reductions. Moreover, the reductions by bin-packing and objective are not easily reproducible by commercial solvers. Since we can presolve the model before solving it with Gurobi, we only use the presolved formulation, except something else is mentioned.

Quality of our problem formulation

We compute a primal solution, if the instance has at least one feasible solution, by a list scheduling heuristic. Since the list scheduling heuristic does not always detect one solution, the order of the tasks is permuted until one feasible solution is found. This solution is used to describe the value **LS** (list scheduling). Then, the LP-relaxation is solved to define the value of LPrelax. After that, we compute the root relaxation and the optimal solution of the instance, if there is one. The root relaxation and the LP relaxation differ in the fact that additional inequalities may be separated in the case of root relaxation. Table 5.4 shows the average gaps if the average is computed for fixed problem sizes, while Table 5.5 shows the average gaps if the average is computed for a fixed objective. Within further discussions of the quality and the performance of our branch-and-bound algorithm, the expression **gap** will be used multiple times. The gap describes the relative distance between the incumbent and the currently best-known dual bound. Using this expression, multiple gaps can be used to analyze the problem formulation. We observe the initial lower and upper bound gap ($gap_{UB} := \frac{|LP-Firstsol|}{FirstSol}$), the root gap ($gap_{LP} := \frac{|LPrelax-Firstsol|}{FirstSol}$), the gap between the best and the first primal solution ($gap_{Firstsol} := \frac{|Best-Firstsol|}{Best}$) and the gap between the root relaxation and the best known primal solution ($gap_{opt} := \frac{|Best-LPrelax|}{LPrelax}$).

Table 5.4 shows the averaged gaps of the different instances. The table shows that for each instance size, the initial gap and the root gap are similar. The considered cutting planes cannot strengthen the LP relaxation significantly. Obviously, some important classes of inequalities describing the facets of the polytope $\mathcal{P}^B$ are missing. However, an optimum solution and the root relaxation have an average gap of 0.022.

Table 5.4 shows that the initial root gap becomes wider when the time window increases; for example, the initial root gap of laXX_s_s is smaller than the gap of laXX_s_l. As the time window increases, the initial root gap also increases. In addition, the gap between two instances with a similar time window setting decreases if the size of the breaks is increased. Thus, considering a fixed time window, the instance with larger ramping durations will have a smaller root gap. A similar tendency of the root gaps can be observed in all columns

Table 5.4: Quality of LP-relaxation, first and optimal solution.

instance	gap_{LP}	gap_{UB}	$gap_{Firstsol}$	gap_{opt}
laXX_s_s	0.166	0.164	0.146	0.015
laXX_s_m	0.222	0.220	0.196	0.019
laXX_s_l	0.275	0.272	0.248	0.018
laXX_s_h	0.391	0.388	0.357	0.021
laXX_m_s	0.087	0.084	0.065	0.018
laXX_m_m	0.186	0.183	0.153	0.025
laXX_m_l	0.246	0.243	0.210	0.026
laXX_m_h	0.329	0.326	0.288	0.027
laXX_l_s	0.048	0.046	0.032	0.014
laXX_l_m	0.099	0.097	0.073	0.021
laXX_l_l	0.182	0.180	0.149	0.026
laXX_l_h	0.232	0.230	0.194	0.028
laXX_r_s	0.066	0.064	0.048	0.015
laXX_r_m	0.165	0.161	0.130	0.026
laXX_r_l	0.261	0.258	0.220	0.025
laXX_r_h	0.309	0.305	0.270	0.028
average	0.204	0.201	0.174	0.0221

of the table. However, the change in the gaps with regard to the initial and optimal solution is weaker than the change in the initial root gap. The averaged root gap and the averaged optimum gap differ by a factor of 10. Thus, our first heuristic solutions are far away from an optimum solution. Therefore, algorithms need to search for near-optimal solutions with a close neighborhood of the LP-relaxation, or best-known dual bound.

This behavior of the gaps is verified concerning the randomized ramping durations within the last four rows. Primarily, the large gaps are associated with objective 8. This is stated by Table 5.5, which shows the gaps when the average is computed concerning the objectives.

Table 5.5: Gaps of the instances broken down by objective functions.

instance	gap_{LP}	gap_{UB}	$gap_{Firstsol}$	gap_{opt}
laXX_0_X_X	0.127	0.125	0.087	0.0349
laXX_1_X_X	0.058	0.059	0.058	0.0011
laXX_7_X_X	0.103	0.102	0.085	0.0165
laXX_8_X_X	0.564	0.557	0.499	0.0388
average	0.213	0.211	0.182	0.0228

Table 5.5 shows that objective 8 leads to large initial and root gaps, while objective 1 leads to small gaps. The small gaps in the case of objective 1 can be explained since the optimization has to decide about the number of standby periods. In case of low energy consumption in the case of machine state standby, the resulting objective value will be near optimal. An expensive primal solution uses a lot of standby, while a near-optimal solution uses as little energy as possible. If the energy consumption for standby is low, then the resulting objective values are similar.

However, the initial gaps are large, and the gap between optimal solution and LP relaxation is up to 4%. Thus, the assumption is verified that our algorithm needs to drive the dual bound to detect the primal solution as early as possible. The root relaxation (rootrelax) does not provide a significantly stronger bound than the LP relaxation, and the additional consideration of valid inequalities does not drive the dual bound. The devised heuristic can explain the relatively large gaps of objective 8. The list scheduling heuristic does not consider the objective, and we take the first primal solution. However, the objective 8 shows that the objective highly influences the scheduling and its performance, and thus, the heuristic efficiency, which works perfectly fine for one objective, can dramatically fail in the case of a completely different objective. Moreover, the objective 8 still shows the property that the optimal solution can be detected within a close neighborhood of the LP relaxation.

In summary, the analysis shows that our algorithm should not stray too far from the best dual bound to compute the optimal solution. However, some diving into branches to detect near-optimal solutions for pruning is tolerable.

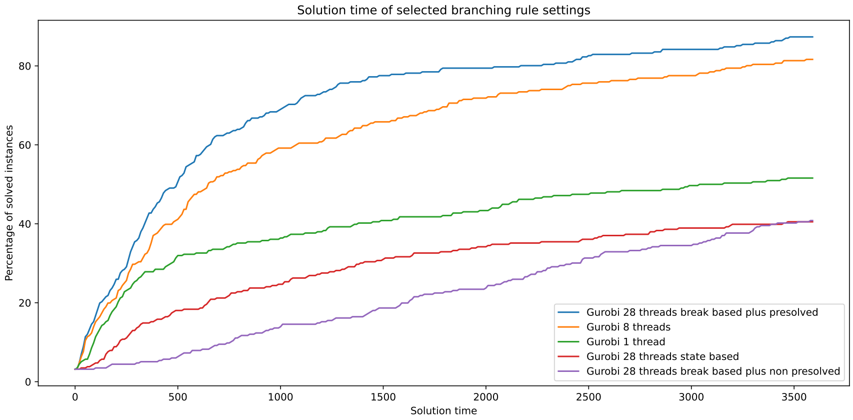


Figure 5.3: Performance of Gurobi using different solver settings and ILP formulations. The solution time is given in seconds.

5.4.2 Analyzing Gurobi's Performance

Figure 5.3 shows the number of instances Gurobi is able to solve within a chosen time limit of 3600 seconds. The perfect result would be if 100% of all instances are solved within 0 seconds. The worst result would be 0% solved instances within 3600 seconds. Figure 5.3 includes five curves. One curve describes the number of solved instances until a specific time for a given setting or formulation. There is a curve to describe the performance of Gurobi solving the formulation (3.1a)–(3.1k) with 28 threads, one curve to visualize the performance of Gurobi solving the instances using formulation (3.10a)–(3.10h) with 28 threads and three curves to describe the performance of Gurobi solving the formulation (3.10a)–(3.10h), which has passed through the implemented presolving, with 1, 8 and 28 threads. A solver setting A solving more instances within a shorter time than a different solver setting B is denoted to have a better performance than a solver setting B. Figure 5.3 shows that the number of solved instances correlates with the number of used threads in the case of solving the presolved formulation. The number of solved instances can be doubled if we apply the implemented presolving before passing the problem to Gurobi. Note that Gurobi is always allowed to apply its presolving. This demonstrates the significant impact of the implemented presolving approach on the resulting solving efficiency of the resulting problem formulation. Gurobi solving the state-based formulation is as efficient as Gurobi using the break-based formulation, which is not presolved by our presented rules. From now on, we call this modeling the non-presolved break-based formulation. This fact impressively shows that the usage of the implemented presolving speeds-up the resulting solution process. Comparing the solution statistics for multi-threaded solving of the presolved break-based formulation, the speed-up is recognizable. If we use a value of 3600 seconds, if an instance is not solved within the time limit, the average solution time of Gurobi with 28 threads on presolved models is 1049 seconds. Without the implemented presolving, Gurobi has an average solution time of 2808 seconds. This leads to a **averaged speed-up of** 2.67. Figure 5.4 visualizes the solution times of two solution approaches for each instance. This figure impressively visualizes that the devised presolved break-based formulation outperforms the state-based formulation. The figure also hints that the total speed-up could be larger than 2.67 since many instances are not solved to optimality and weaken the speed-up factor by their value of 3600 seconds. No instance is solved faster using the non-presolved break-based formulation than by using the presolved break-based formulation. Moreover, one can see that if the solution process using the presolved break-based formulation needs a certain amount of time, the solution approach usage of the non-presolved break-based formulation takes even longer.

The number of solved instances over time by usage of the presolved break-based formulation using 8 or 28 threads is similar. The difference between the parallelized solution processes using 28 or 8 threads is less significant but present. The averaged solution times are 1049 and 1331 seconds. Thus, the averaged speed-up by 28 threads is 1.27. Thus, the solution process using 8 threads and the presolved break-based formulation also outperforms the solution process using 28 threads and the state-based formulation.

Figure 5.5 shows the required solution times of Gurobi using 8 threads and Gurobi using 28 threads solving the presolved break-based formulation. Figure 5.5 shows the

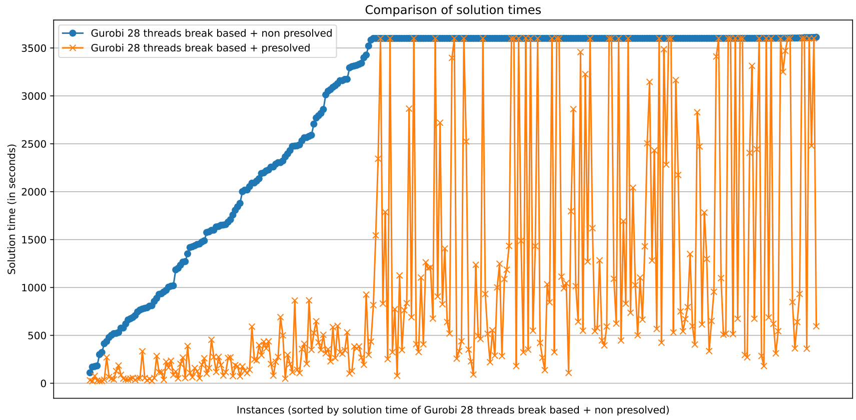


Figure 5.4: Results of Gurobi, using 28 threads and non presolved, are sorted from small to large solution time. In addition, the solution times of Gurobi using 28 thread and the break-based formulation using the presolved formulation are also plotted using the same order of the instances. Visualization of the solution time (in seconds) of solving the instance with Gurobi and 28 threads and the break-based formulation. This figure shows that all the presolved formulations can be solved faster than the non-presolved formulation.

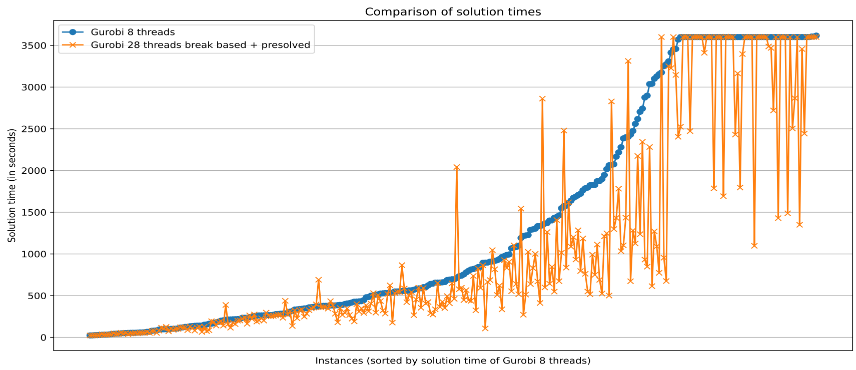


Figure 5.5: Results of Gurobi using eight threads, are sorted from small to large solution time. In addition, the solution times of Gurobi using 28 thread and the break-based formulation using the presolved formulation are also plotted using the same order of the instances.

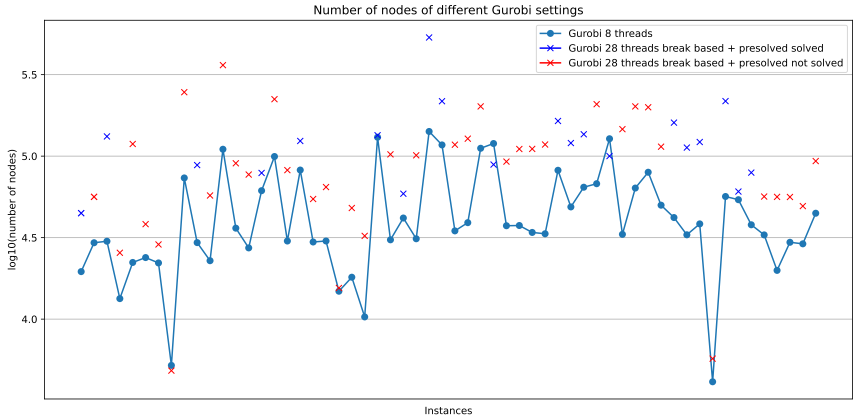


Figure 5.6: This figure displays the number of nodes visited by GUROBI using eight threads solving the presolved break-based model. In addition, the corresponding number of visited nodes by GUROBI using 28 threads are displayed marked as solved or not solved. Only the instances are considered, which cannot be solved within the time limit.

required solution times of GUROBI using 8 threads and GUROBI using 28 threads solving the presolved break-based formulation.

The curves show that a larger number of threads and leads to the possibility of exploring more nodes per second. First of all, GUROBI using 8 threads can solve certain instances faster than GUROBI using 28 threads. This can happen when GUROBI internally weights the branching behavior or the calls of the heuristics differently if more threads are available. However, the possibility of solving more nodes per second substantiates the performance relation of both solver settings.

Now, we want to analyze whether the possibility of using 28 threads results in large branch-and-bound trees. In Figure 5.6, the instances where GUROBI failed to compute the optimal solution with 8 threads are considered, and the number of visited nodes is displayed. In addition, the number of visited nodes of GUROBI using 28 threads is shown in combination within a marker, whether more branch-and-bound nodes were necessary to solve the problem to optimality or not. Figure 5.6 shows that GUROBI using 28 threads mainly visits more nodes than GUROBI using 8 threads if GUROBI using eight threads is not able to solve the instance to optimality. GUROBI using 28 threads uses not more than $\sqrt{10}$ times more nodes than GUROBI using 8 threads, if optimality is not proven by GUROBI using 8 threads. If we also consider GUROBI solving the presolved break-based formulation single-threaded, the required number of nodes looks similar.

However, the results of GUROBI solving the state-based model and GUROBI solving the break-based model using 28 threads show that the state-based formulation solves more instances within a shorter period. Figure 5.7 shows that there is no rule describing whether GUROBI solves instances faster by using the state-based formulation or the break-based formulation. There are instances solved faster with one of the formulations and slower or even not at all solved by the other possibility. Since the instances are sorted by their solution time of one solver setting, the plot seems very chaotic.

Figure 5.8 shows that the state-based formulation can solve a small subset of instances faster. These instances have the objective of minimizing energy consumption in common. Otherwise, the break-based formulation is faster. The objective _1_ leads to a different behavior since this objective allows multiple optimal solutions differing by processing starts.

In summary, the implemented presolving leads to a significant speed-up of the solution process of GUROBI solving the break-based formulation. The formulation outperforms the existing state-based formulation. However, the number of required branch-and-bound is too large to be considered to solve larger instances.

5.4.3 Analysis of the Implemented Algorithms

This thesis includes algorithmic approaches improving the solution process by presolving and propagation, branching, valid constraints and column generation from Section 4.1 to Section 4.4. This subsection considers the impact of these techniques on the solution process. Figure 5.9 shows the percentage of solved instances until a certain time limit (3600 seconds). Figure 5.9 displays seven different curves, representing the solution process im-

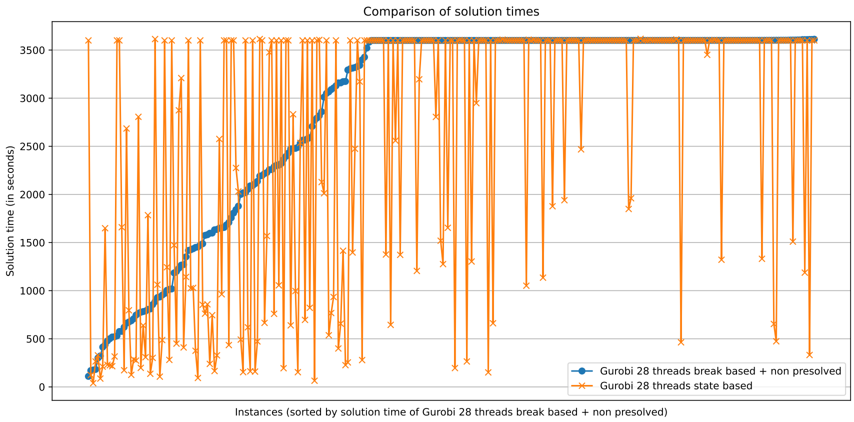


Figure 5.7: Comparison of GUROBI solving the state-based formulation and GUROBI solving the break-based formulation. The Results of GUROBI using 28 threads and the break-based formulation, are sorted from small to large solution time. In addition, the solution times of GUROBI using 28 thread and the state-based formulation using the formulation are also plotted using the same order of the instances.

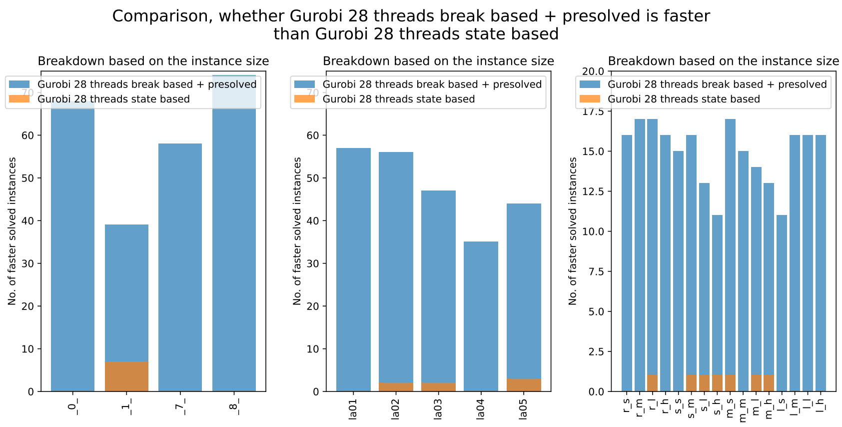


Figure 5.8: Statistics showing whether the state-based or the presolved break-based formulation leads faster to the optimum solution.

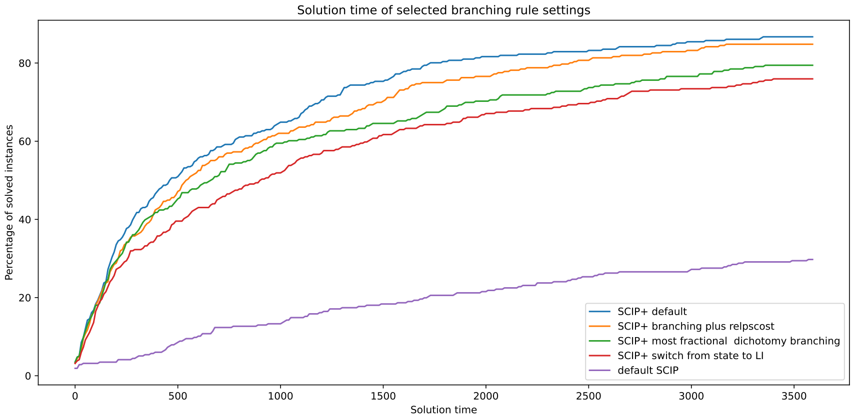


Figure 5.9: Solution times of Scip+ with using branching rule combinations. The solution time is given in seconds. In addition, the best run of Scip in default settings is visualized.

plemented in Scip using different combinations of branching rules. One curve, belonging to *default Scip*, is outperformed by each further displayed run. Additionally, one can see that *most infeasible branching* is not a meaningful choice. The usage of workload branching (4.40), (4.39) has a significant impact on the solution time. The branch-and-bound algorithms that use the workload branching are able to solve at least 60% of all instances within the chosen time limit. Different selections of the branching rule, which is in charge of producing the integrality of the task variables, vary the solution times. Using the most infeasible branching rule instead of an assignment constraint branching 4.43 is the worst approach of our implementations. This is reasoned to the fact that most infeasible branching does not reuse information from the underlying scheduling problem to perform the branching. The usage of the assignment branching by the dichotomy approach, mentioned by van den Akker, also leads to a slight change in performance. This is reasoned by the fact that the dichotomy is not the most important substructure of fractional solutions. The consideration of problem-specific substructures as the workload is more important to decide about useful branching candidates.

One surprising result is that the performance of the reliable pseudo-cost branching in combination with workload branching performs well. The reason is that reliable pseudo-cost branching works well in cases where the time window for processing the tasks is nearly fixed by comparing the objective coefficients. The workload branching adjusts the time windows in a way that only the integrality of the tasks must be created. Thus, this branching is a considerable selection.

One disappointing result is the solution times of the hybrid workload branching and assignment constraint branching rule. This rule is considered to automatically detect whether to perform a workload branching or to perform an assignment constraint branch. However, this branching fails to be as successful as best-devised branching. Thus, there is some space for improvement. However, the solution times are still outperforming the default settings of Scip. Note that Scip is not explicitly trained to solve the considered schedule problem.

The detailed analysis of some selected branching statistics shows that a fixed default strategy is possible. However, there are upwards and downwards outliers.

Next, we consider the best runs using Scip+ with the devised branching rules. We present results visualizing that small changes in the choice of the branching candidate selection influence the result of the implementation: Figure 5.11 shows that small changes within the branching period selection of the workload branching can increase or decrease the solution time. But the curve stays nearly equal. Thus, we do not expect significant changes from the implemented rules if the optimal parameter choices are known. These results show that there is space for improvement of the branch-and-bound algorithm. There is a lack of a control system that leads to better problem-adapted branching rules.

In addition, we also see changes within the assignment constraint branching lead to only small changes within the solution times. Figure 5.12 shows that switches to different assignment constraint branching can decrease or increase the solution time. The resulting curve of the sorted solution time looks similar. This figure also shows that the analysis for further strategies in the case of assignment constraint branching was necessary and has led to better results.

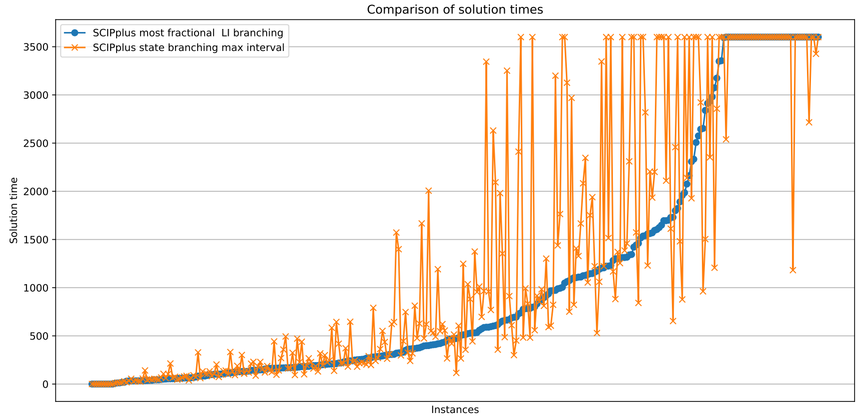


Figure 5.10: Detailed visualization of the solution times by Scip+ using different settings. The solution time is given in seconds.

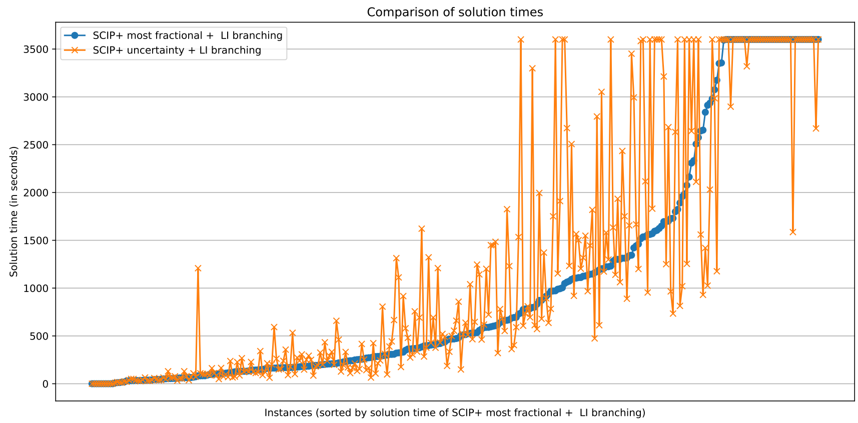


Figure 5.11: Detailed visualization of the solution times by Scip+ using different settings. The solution time is given in seconds.

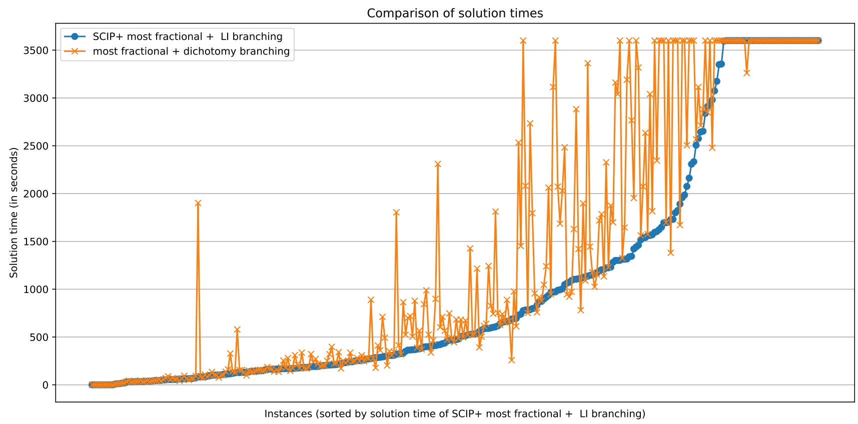


Figure 5.12: Detailed visualization of the time-consuming runs of Scip+. The solution time is given in seconds.

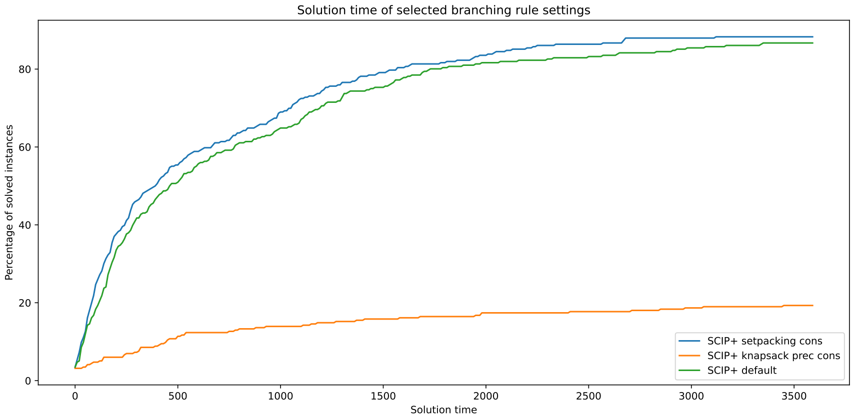


Figure 5.13: Solving time of instances, which are solved by SCIP+ with different handling of the precedence constraints. The solution time is given in seconds. The curve 'setppc' describes the solution times of the instances when the problem formulation with the precedence constraints is created as setpacking-constraints.

Figure 5.13 visualizes the different variants of considering precedence constraints. Figure 5.13 includes three curves: one curve visualizing the number of solved instances over time when using the aggregated precedence constraints (aggregated), one curve displaying the number of solved instances when using a description using set packing constraints (setppc), and one curve displaying the number of solved instances when using a self-implemented separator for precedence constraints (separator). The curve separator uses the aggregated constraints to check the feasibility of solutions, and the separator separates disaggregated precedence constraints.

Figure 5.13 shows that SCIP using the set packing constraints as precedence constraints describes the best approach. Further, the solution approach only using the aggregated precedence constraints performs not well. The run with the precedence constraints (4.2) additionally includes a higher frequency of separating knapsack constraints and additional computation of Gomory cuts and cuts from aggregation. However, the solution process cannot reproduce the information present in the disaggregated precedence constraint formulations. We implemented a constraint handler for precedence constraints since most of the precedence constraints are not necessary within the solution process. However, we could not reproduce the same strength of presolving and conflict generation as the constraint handler of set packing constraints. Thus, the runs with simple set packing constraints seem to be more efficient than the run with real precedence constraints. However, both runs are acceptable and solve over 80% of all problem instances. This run is stated here as "separated" and includes the aggregated precedence constraints to check the feasibility of primal solutions and methods to separate disaggregated precedence constraints and detect valid precedence constraints. The difference between the run "separated" and the run "setppc" can be justified by the additional time required for searching for valid further precedence constraints. The search for valid precedence constraints requires about $\frac{2}{3}$ of the total consumed time of the precedence constraint handler. Considering this waste of time and the separation of additional precedence constraints reason for the small shift within the curves. However, the run "separated" initially uses fewer constraints and additionally needs to spend time within the computation of further valid precedence constraints. Thus, the shift between the run "separated" and "setppc" can be justified.

Figure 5.14 shows that solution approaches using the disaggregated precedence constraints either by set packing constraints or by the implemented separator solve a similar number of problems per instance type (la01, la02, la03, la04, la05). Figure 5.14 clearly shows that SCIP using the default settings and set packing constraints results in similar statistics as SCIP+ using the aggregated precedence constraints. The default settings of SCIP do not separate set packing constraints within the branch-and-bound tree. Thus, the precedence constraints are not separated for each branch-and-bound node in depth larger than one. Therefore, the pure existence of precedence constraints and the corresponding information in presolving is not sufficient to improve the solution process. It is crucial to separate these constraints.

Figure 5.15 shows selected solver settings and the number of solved problems of the different underlying scheduling problems. One can see that the SCIP implementations equally

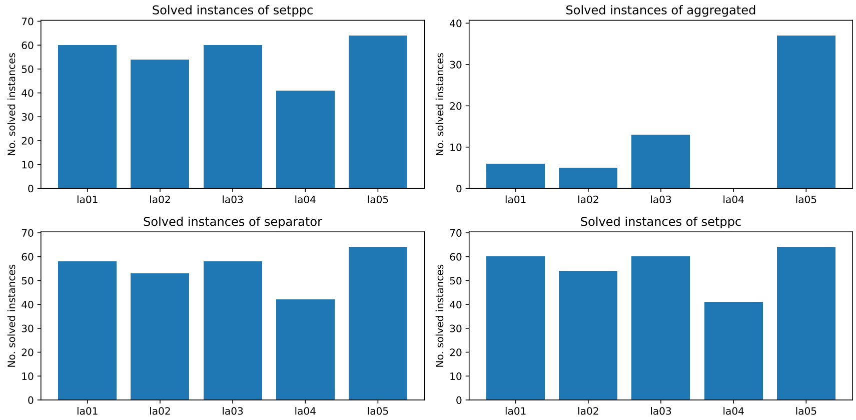


Figure 5.14: Visualization of the different solver settings and their performance on all instances for time windows and ramping durations.

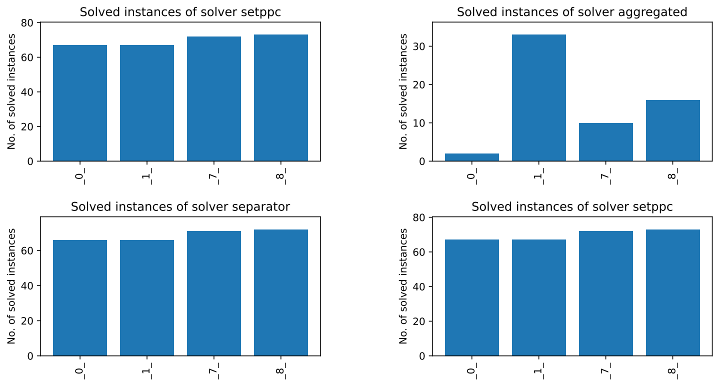


Figure 5.15: Visualization of the different solver settings and their performance on all instances for different objective functions.

struggle with each type of scheduling problem. The noise within the data is based by difficulties in detecting the optimal solution or driving the dual bound for special instances. Figure 5.15 visualizes the number of solved instances broken done by the type of objective. One can see that there is no implication that some objective should be solved by one specific type of precedence constraint setting. In contrast, the problem size clearly shows that the usage of disaggregated precedence constraints, which are separated within the branch-and-bound tree as set packing constraints or by a separator, is preferable. This is displayed in Figure 5.16. In summary, the usage of aggregated precedence constraints performs badly if the disaggregated precedence constraints are not separated additionally. Then, there is no difference in whether new precedence constraints are detected within the branch bound or not. Moreover, the initial generation of information from the disaggregated precedence constraints is a crucial part of the solution process.

Now, different components of the implementation are disabled such that the performance loss is displayed. Figure 5.17 shows that slight changes within the algorithm directly lead to loss of performance. Figure 5.17 shows a curve "SCIP+ default" which describes the best-performing branch-and-bound algorithm next to GUROBI using 28 threads. Instead of solving the instances by using of the reference implementation, the breaks can be generated by column generation. Column generation requires that many further valid inequalities are not valid since the consideration of cutting planes in column generation is complicated and hard to implement (but possible). The missing valid inequalities from GUB cover cuts and clique cuts cause the corresponding loss of performance.

The most obvious loss of performance was reached by disabling the state-constraint branching. A factor of 2 gives the relation between the number of solved instances. Since the state-constraint branching is designed to be dual-bound driving, while the assignment constraint branching is designed to create primal solutions, the branch-and-bound tree

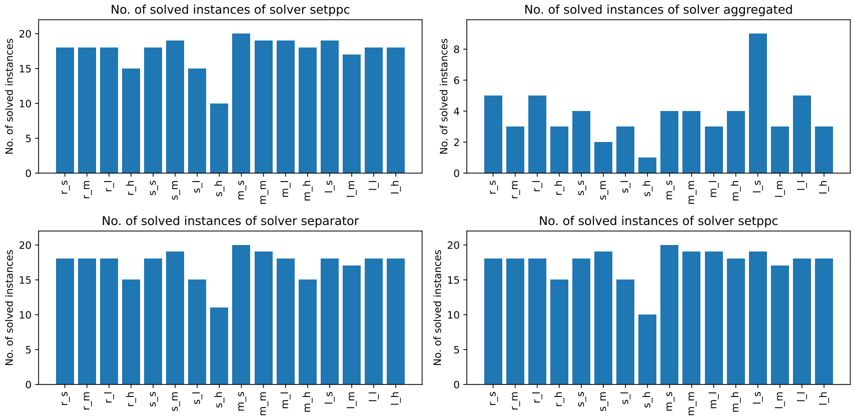


Figure 5.16: Visualization of the precedence settings number of instances which are not solved.

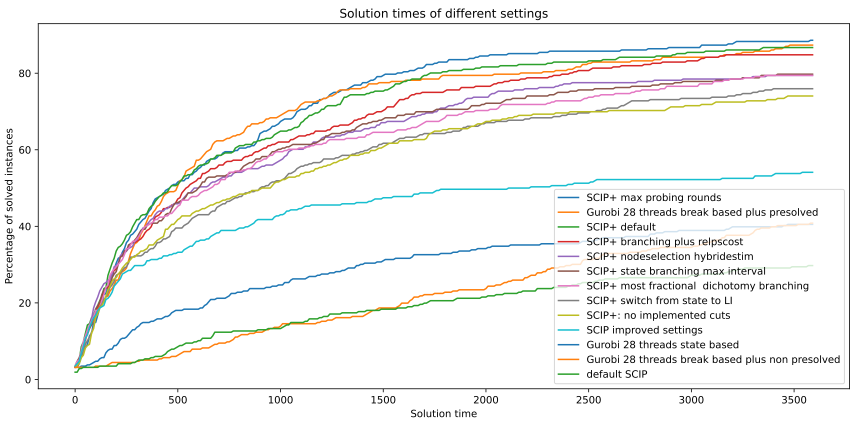


Figure 5.17: Visualization of the effect of the different algorithms and parts of the solver. The solution time is given in seconds.

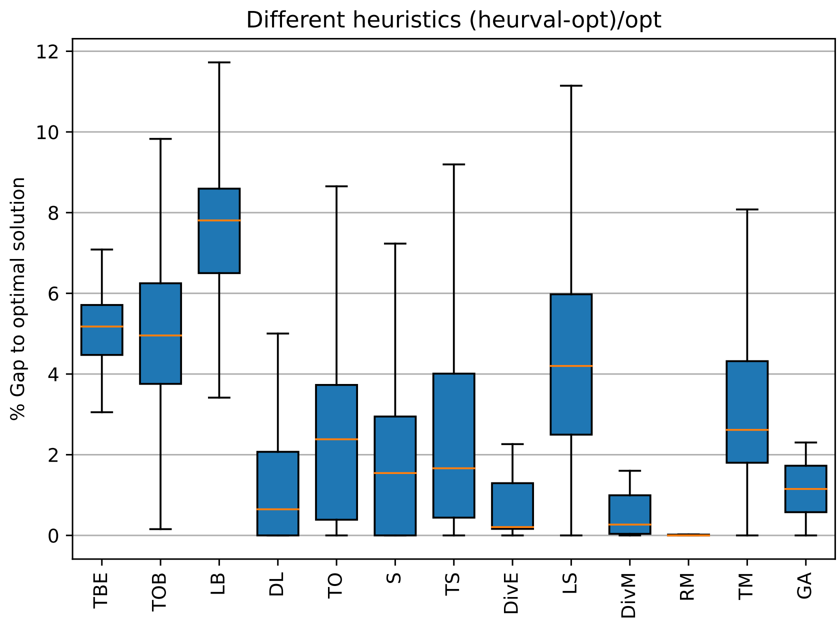


Figure 5.18: The average gap of the implemented heuristics. The figure shows list scheduling heuristics using backward scheduling by due dates (TBE), backward list scheduling by information from the fractional solution (TOB), neighborhood exploration by dynamic programming, dynamic programming (DL), forward list scheduling using the information from the fractional solution (TO), shifting the current best solution in time (S), list scheduling by using LIFO-rule from the release dates, diving on variables matching the expected value (DivE), classical LIFO list scheduling (LS), diving using the maximum fixation (DivM), rounding of the variable with the largest (fractional) value (RM), and a genetic algorithm (GA).

is misleading. Thus, many instances are not solved to optimality. The disabling of the implemented heuristics also leads to a loss in performance. This is reasoned by the fact that the devised heuristics are designed to be applied early and compute near-optimal solutions. If these heuristics are missing, the MILP-heuristics of SCIP need to compute primal solutions. However, only ALNS and rounding heuristics compute solutions. Thus, more branch-and-bound nodes need to be created until the optimum solution is detected.

Changing the branching rule to create schedules can also increase and decrease the performance of the solution algorithm. Also, the disabling of the propagation algorithm leads to a loss of performance. Obviously, the disabling leads to larger problems that need to be solved at each branch-and-bound node. However, the number of reductions is not large enough to significantly increase or decrease the number of solved problems.

In summary, the algorithm is able to solve about 70% of the provided test instances within one hour.

The Implemented Heuristics

Our implemented heuristics have a different performance and usage. Most of the implemented heuristics are considered to compute initial solutions. Further heuristics are devised to explore the neighborhood of the current best solution.

Figure 5.18 shows that all of our implemented algorithms provide an optimum gap of 0% to 12%. The lines denote the outliers of the objectives computed by the heuristics. The orange line describes the average value, and the blue box denotes the 75% of all computed solutions. This figure shows that our diving heuristics provide near-optimum solutions, while the large neighborhood solutions suffer from computing many bad solutions. The list scheduling heuristics provide a large range of solutions since these heuristics can be applied multiple times within the branch-and-bound node. Figure 5.19 shows that most of the initial solutions are computed by our diving heuristics. Different heuristics of SCIP are also able to compute initial solutions. Furthermore, in combination with 5.18, our initial solution has a gap of less than 2%. Then, the other heuristics very often cannot compute better primal solutions since we are already at a near-optimal solution. Figure 5.20 is created by tracking the algorithms computing the optimal solutions and shows that most of the optimal solutions are computed by evaluating the LP relaxation at branch-and-bound nodes. While most of the initial solutions are computed by our problem-specific heuristics,

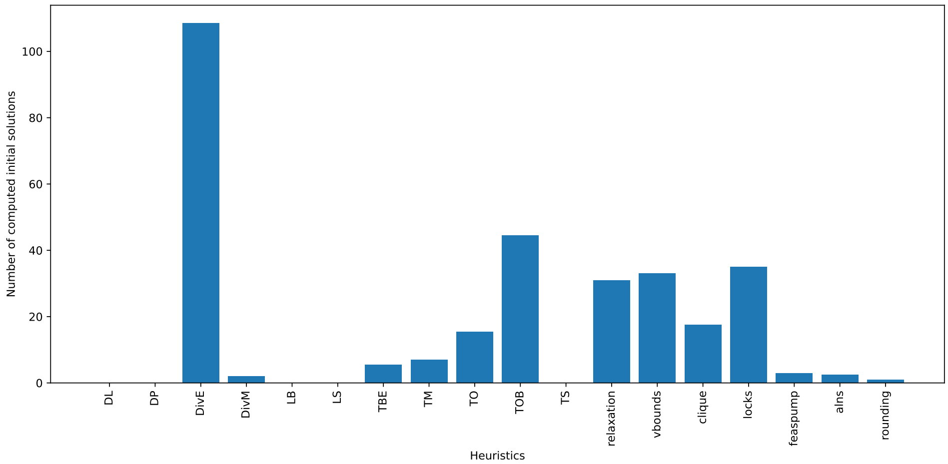


Figure 5.19: The average number of computed initial solutions by the different used heuristics. The figure shows the large neighborhood approach by dynamic programming approach (DL), dynamic programming (DP), diving using the distance to the expected value (DivE), diving using the maximum value (DivM), backward (LB) and forward large neighborhood by list scheduling (LS), list scheduling heuristics using backward scheduling by due dates (TBE), backward list scheduling by information from the fractional solution (TOB), and different heuristics implemented in SCIP .

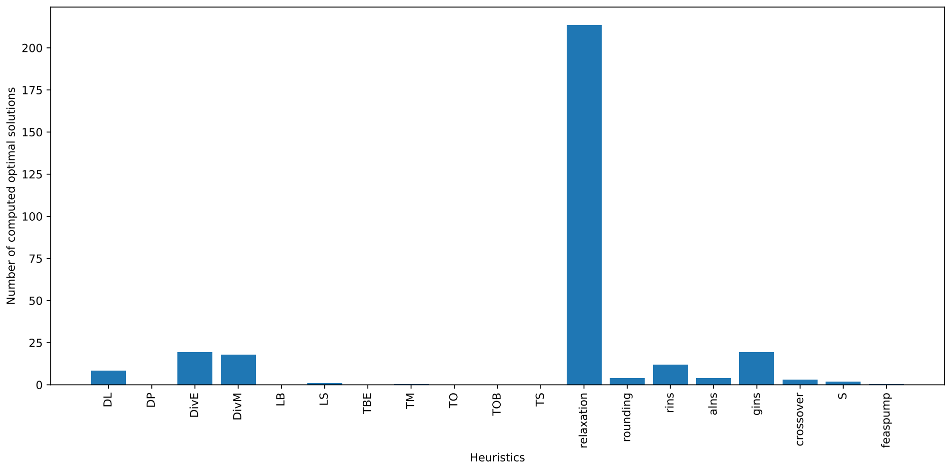


Figure 5.20: The average number of computed optimal solutions by the different used heuristics. The figure shows the large neighborhood approach by dynamic programming approach (DL), dynamic programming (DP), diving using the distance to the expected value (DivE), diving using the maximum value (DivM), backward (LB) and forward large neighborhood by list scheduling (LS), list scheduling heuristics using backward scheduling by due dates (TBE), backward list scheduling by information from the fractional solution (TOB), and different heuristics implemented in SCIP .

the optimum solution can be computed by LP-relaxations. Further optimum solutions are computed by our large neighborhood approach using dynamic programming and diving heuristics. In addition, classical Scip heuristics compute some optimum solutions. In addition, the list scheduling heuristics are too often not able to compute the optimum solution. However, the heuristics are able to compute multiple primal solutions. They help to prune the branch and bound tree initially and thus speed up the finding of the optimal solution. But these visualizations allow for the criticism that the implemented heuristics rarely find an optimal solution. This reveals potential for further improvements to the algorithm

Comparison of Different Separation Strategies

The consideration of different separation algorithms and different settings is only minor since our proposed cutting planes are not a major part of the solution process. Although we analyzed the cutting planes, the number of separated inequalities is small. This fact is visualized in Figure 5.21.

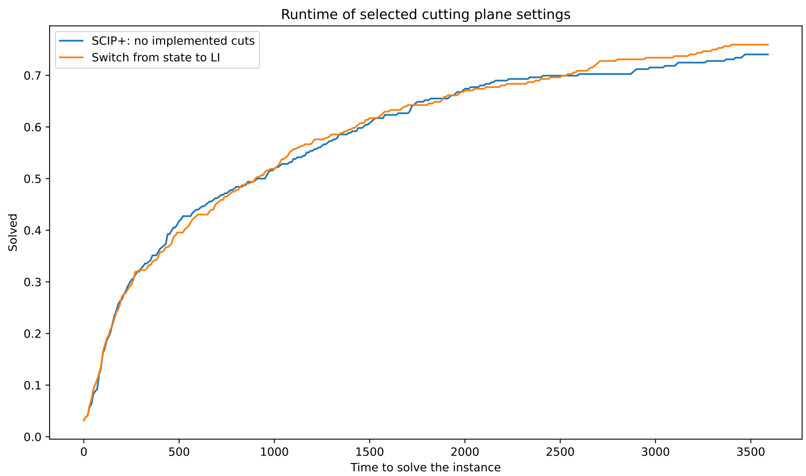


Figure 5.21: Visualization of the impact of the cutting planes. This figure shows a curve, visualizing the solution time using our cutting planes (Switch from state to LI) and a curve using the same algorithms, but the separation of our implemented cutting planes is deactivated (no implemented cuts). The solution time is given in seconds.

Comparisons of Gurobi And the Preferred Scip + Run

Within this section, we compare Gurobi with 28 threads with the implemented single-threaded implementation of Scip+, which uses Gurobi as an LP-solver. Both solvers solve a similar number of instances. The Figure 5.22 shows that the solved instances differ. If one instance is solved by Gurobi, there exists the possibility that Scip+ cannot solve the instance within 3600 seconds. In addition, there also exists instances where Scip+ can solve instances that Gurobi cannot solve. The two curves do not resemble one another. However, some instances are solved by both solvers nearly instantly.

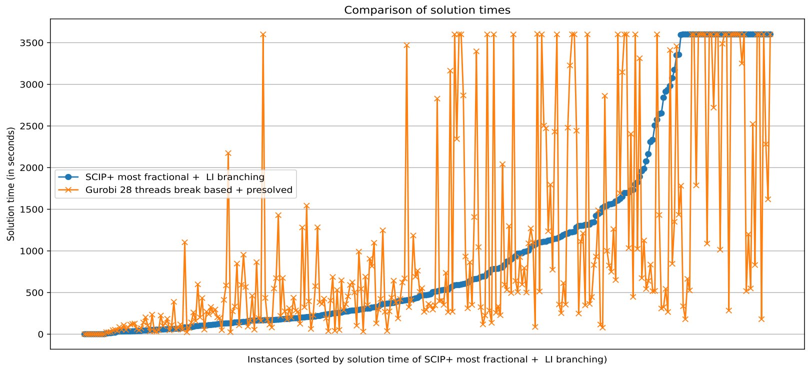


Figure 5.22: Detailed visualization of the time consuming runs of Scip+. In addition, the solution times of Gurobi are visible.

The average solution time of Gurobi with 28 threads is 1077 seconds. The average solution time of Scip+ is 1084 seconds. Note that the time limit is used while computing the average solution time. When considering only the instances solved by both solvers, the average solution time is 654 seconds for Gurobi and Scip+ 641 seconds. Therefore, the solution time of the solvers is similar. Note that Gurobi is allowed to use 28 threads and uses the presolved break-based formulation while Scip+ is single-threaded. In contrast, Scip uses problem-adapted solution strategies while Gurobi is used as a black-box MILP solver.

Since Gurobi using 28 threads and Scip+ provide a similar performance when considering the solution times, further comparisons are therefore needed.

The solver Gurobi can solve some instances at the root node, while Scip+ requires some nodes to compute the optimum solution. Thus, the distribution of the required number of nodes to prove optimality is discussed. The distribution of the required number of nodes to solve the instances is visualized in Figure 5.23.

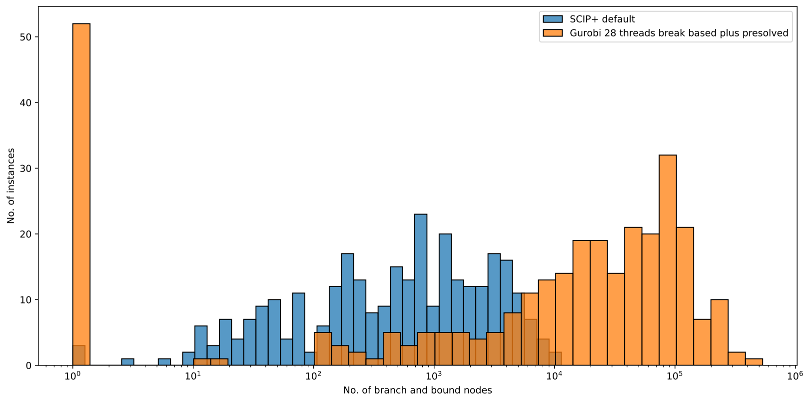


Figure 5.23: Visualization of the required number of nodes of Gurobi using 28 threads with Scip+. If a solver does not find the optimum solution, the number of visited nodes is displayed.

Figure 5.23 shows that the distribution of the required number of nodes to solve the problem is spread from 1 to 10^5 for Gurobi using up to 28 threads and from 1 to $5 \cdot 10^3$ for Scip+. A similar result can be seen when considering Gurobi using up to 8 threads and Scip+. Thus, the problem-specific branching reduces the required number of nodes significantly. The number of nodes of Gurobi with 8 and Gurobi with 28 threads seems to be equal. However, the number of solved instances is smaller in the case of Gurobi with 28 threads. One can suspect that the Gurobi always requires more branch-and-bound nodes than Scip+ since the blue bars are shifted to the right of the orange bars of Scip+.

Figure 5.23 shows that the number of solved instances at the root node is larger in the case of Gurobi. Thus, we need to watch the solution times of those instances, which can be solved at the root node by Gurobi. Figure 5.24 shows the solution times for the instances, which Gurobi solves in the root node. In the case of Gurobi and the case of Scip+, most instances are solved in the range 100 to 1000 seconds. Some instances are not solved by Scip+. These instances use the energy demand objective (_1_), and the optimal

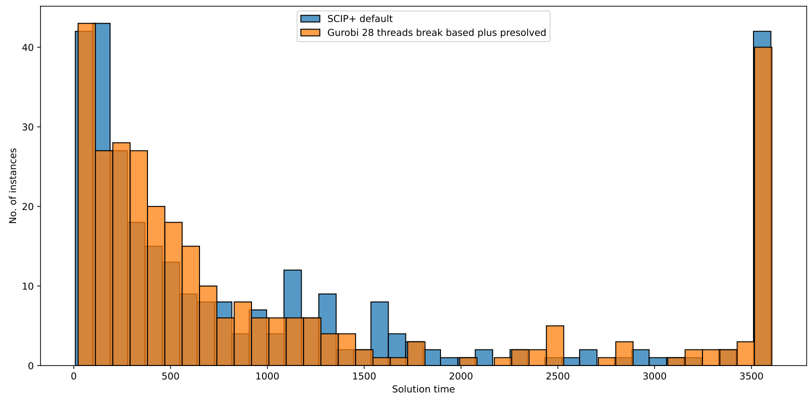


Figure 5.24: Solution times of instances of Gurobi using 28 threads and the presolved break-based formulation and Scip+.

objective value equals root-relaxation. Using heuristics and cutting planes helps Gurobi to find the optimal solution at the root node, while the settings of Scip+ are chosen in the way that these heuristics are only applied at specific depths of the branch-and-bound tree. Thus, Gurobi uses fewer nodes in the case of the considered instances. Neglecting those solved instances, the required solution times of Gurobi and Scip are similar.

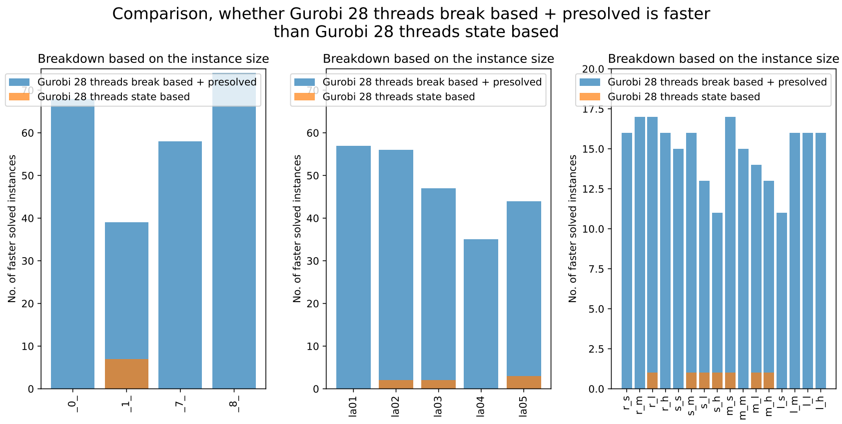


Figure 5.25: Visualization of the different solution times of Gurobi 28 and Scip+, if their solution time differs in at least 1000 seconds, or one solver does not solve the problem within 3600 seconds.

Figure 5.25 illustrates how the objective influences the outcome, whether the implemented branching algorithm will be more or less successful than the commercial branch-and-bound algorithm provided by GurobiẆhile objective 0,1 no rule can be derived, the implementation should be used for objective 7 and the commercial branch-and-bound algorithm for objective 8. The difficulty of objective 8 is that the dual bound does not grow as fast as in the case of objectives 0,1 and 7 since there are negative energy prices. Thus, the fixation of the machine state is not as successful in that case as it would be with strict positive energy prices.

Analysis of a Problematic Instance

Gurobi can solve the instance la02_8_r_h with 28 threads, and it can run single-threaded in at most 1600 seconds. In contrast, Scip+ cannot solve the instance within 3600 seconds. Attempting to use reliability pseudo cost branching instead of assignment constraint branching in the Scip+ implementation to schedule the tasks after fixing the workload was unsuccessful, but the instances can be solved by using classical variable branching rules. However, skipping the workload branching in the case when we forecast an unsuccessful branching and using reliability pseudo cost branching is one efficient way to work around it.

The instances not solved by Scip share a common property: the machine profile is

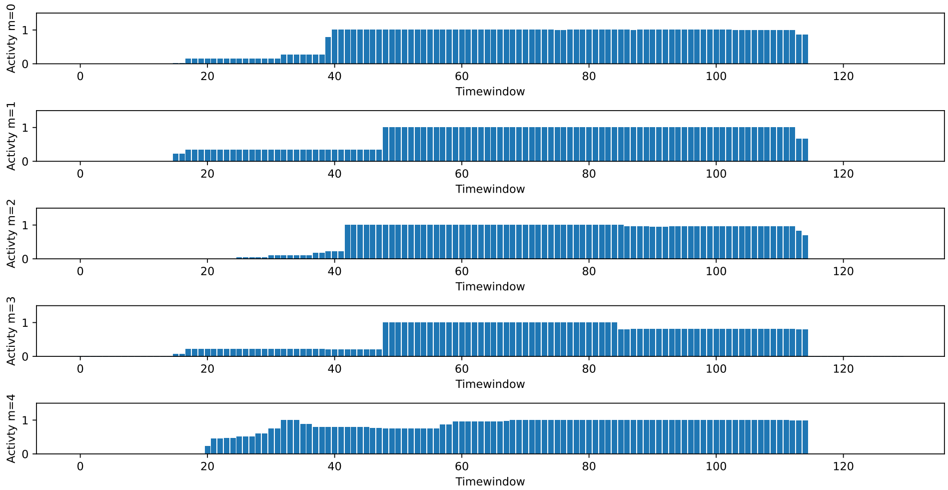


Figure 5.26: Workloads of the root relaxation of $la01\_0\_s\_s$.

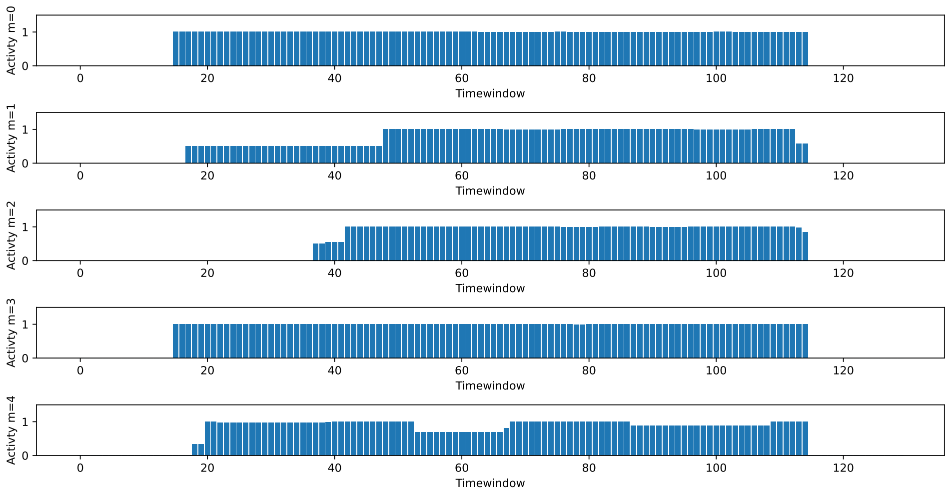


Figure 5.27: Workload of relaxation of $la01\_0\_s\_s$ after 15 branchings.

nearly integral but not completely. Thus, the workload branching can detect a branching candidate. However, the development of the dual bound after branching by workload branching Therefore, we provide a second run only using reliability branching and a third run only using assignment constraint branching if we detect the structure of a nearly integral workload. These instances show that switching from state-constraint and assignment-constraint branching can become crucial. However, one could also use reliability branching to solve the instances successfully.

The obvious question at this point is: **what is the difference of fractional solutions of objective 8 and 0?**

To answer this question, two root relaxations of the instances $la\_02\_8\_r\_s$ and $la01\_0\_s\_s$ are provided. The Figures 5.26, 5.27, 5.28, and 5.29 show the fractional solution at the root node and a fractional solution within the branch-and-bound tree, at a node of depth of 10 of the instance $la01\_0\_s\_s$ and in depth of 15 in the case of instance $la\_02\_8\_r\_s$. Each of the figures consists of 5 sub-figures. Each subfigure, indexed by 0 to 4 are associated to machine $m \in \{0, \ldots, 4\}$. The x-axis denotes the discretized time window, and each subplot shows the machines' workload per period. The instances are presented in A.2.2 and A.2.3.

The instance $la01\_0\_s\_s$ is solved quickly by SCIP+, while the instance $la02\_8\_r\_s$ requires more than 3600 seconds to be solved by SCIP+. One can see that the workload

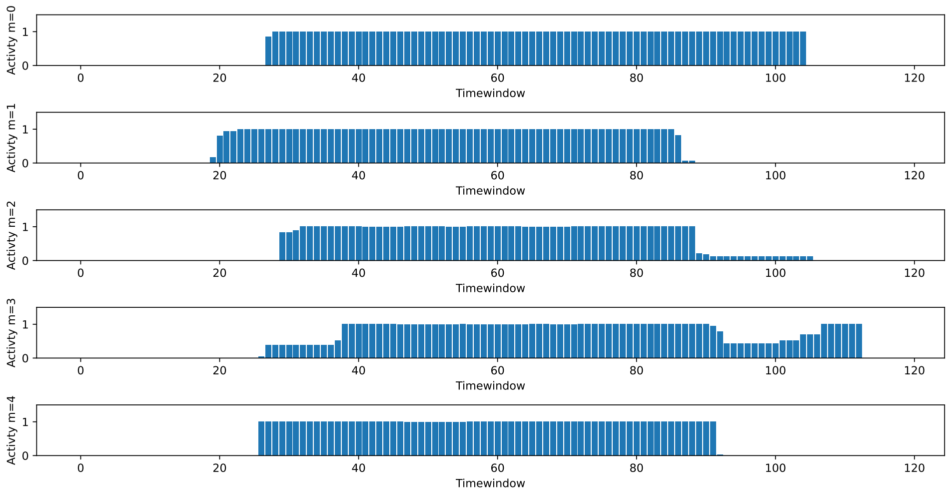


Figure 5.28: Workloads of the root relaxation of $la02\_8\_r\_s$.

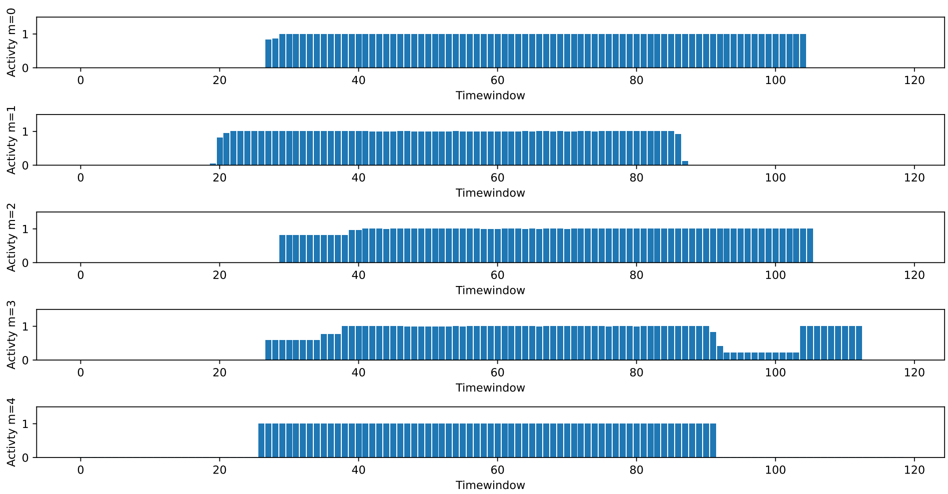


Figure 5.29: Workloads of the root relaxation of $la02\_8\_r\_s$ after 10 branchings.

of $la01\_0\_s\_s$ is more fractional than the workload $la02\_8\_r\_s$ in the root node. The instance $la01\_0\_s\_s$ forces the devised branch-and-bound algorithm to enforce the integrality of the workload on each machine. This enforcement changes the workload profile in a significant way. Thus, the objective and the dual bound are increased while performing these branchings.

In contrast, the root relaxation of instance $la02\_8\_r\_s$ has multiple machines with nearly integral workload. The tasks that are assigned to a machine with an almost integer workload are also processed within a defined range. Extending the range is expensive. The parameters of the machines show that the instance $la02\_8\_r\_s$ has energy-efficient machines and machines with high energy demand. This contrast of the machines within the same instance can lead to the case that the machine with the energy-efficient energy demands is used to absorb most of the changes due to branching rules. Another point of the fractional solution is, in the case of $la02\_8\_r\_s$ is mostly located on one machine, the machine with the energy efficient demands.

It is, therefore, necessary to decide which branching rule to use based on the fractional solution, on the parameters of the instance, and on the properties of the fractional solution. A simple selection rule considering a prioritization of the branching rules is not sufficient. A selection rule has been implemented, but it is not strong enough to outperform the prioritization rule. Even more detailed analyses are important so that the correct branching rule is selected before the resulting branching.

5.4.4 Analysis of the Column Generation Approach

Using the column generation algorithm to solve the LP relaxation at each branch and the bound node is not helpful within the considered problem sizes. However, we provide an analysis of the number of generated variables. The number of variables after presolving is small enough to solve the resulting LP-relaxations efficiently and the advantage of the column generation algorithm is not present.

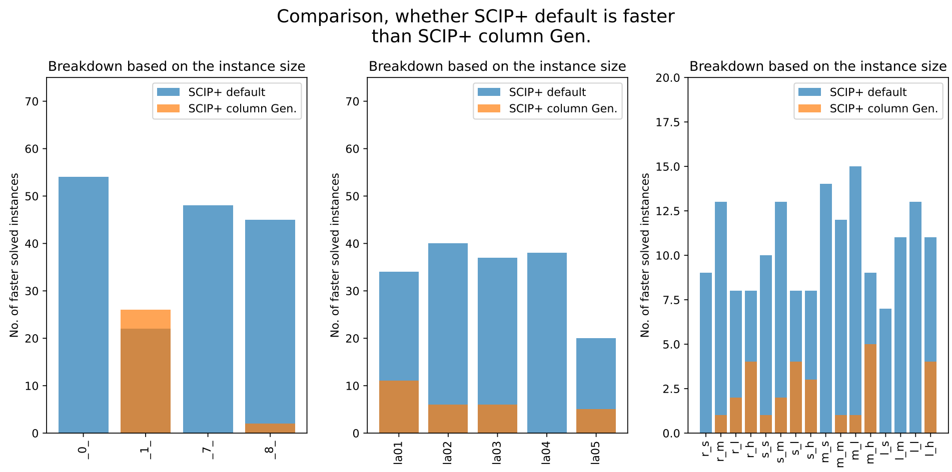


Figure 5.30: The visualization illustrates how many instances were solved more rapidly by both Scip+ using the column generation method and Scip+ employing the Most-Fractional and LI Branching approach. The height of the bars represents the number of instances successfully addressed by each respective solving strategy.

Figure 5.30 shows that the branch-and-bound and cut approach outperforms the branch and price approach. This is reasoned by the fact that the implemented presolving, in combination with the MILP-based presolving of Scip can reduce the problem to a similar size, which also can be achieved by column generation. Thus, the advantage of solving smaller problems at the branch and price nodes is gone. Moreover, the branch and price approach is not allowed to separate the same cutting planes as the branch-and-bound process since these would change the pricing problem. Thus, the dual bound increases faster in the case of the branch-and-bound process. Therefore, most of the problems are solved more efficiently by the branch-and-bound approach. However, some instances are solved faster by the column generation approach than by the branch-and-bound approach. Mostly, these are instances with objective _1_. These instances suffer from the fact that the optimal primal solution is very close to the root solution. If multiple breaks cannot be

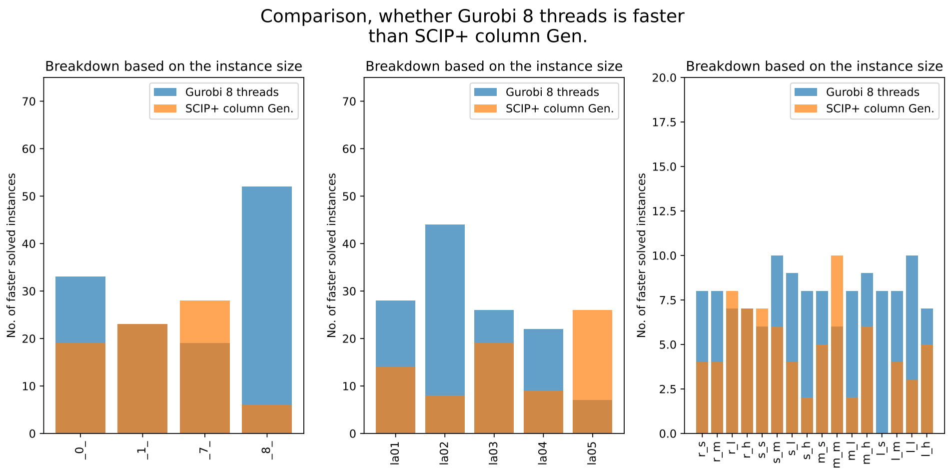


Figure 5.31: Visualization of the number of instances by GUROBI using eight threads and by the column generation approach. The plots in the first row show the number of instances that are solved faster by the column generation approach. The second row shows the number of instances that are solved faster by GUROBI using 8 threads. If the solution time of some instances differs in less than 100 seconds, the solution time is considered to be similar.

used in optimal solutions, the heuristics still can try to use them. However, we are confident that the approach will better unfold its strengths on larger instances by better utilizing the advantages of its structure and methodology. Thus, the heuristics are guided to find the optimal primal solution because the necessary variables are present and additional, non-necessary variables are often absent. Therefore, some instances can be solved faster than by the devised branch-and-bound approach.

The branch and price approach works as well as GUROBI using the already presolved formulation and up to 8 threads. Figure 5.31 shows that the column generation approach is able to solve some instances faster than GUROBI using up to 8 threads and faster than SCIP+. Some instances, which require the computation of further cutting planes to strengthen the dual bound, are solved faster by GUROBI than by the column generation approach, for example, the instances with size $\_l\_s$. There, the breaks are large, and the time window is small. For those instances, the time window is as the schedule is nearly fixed, and the remaining problem is a scheduling problem. Initially, the number of breaks is small, and the column generation approach has no advantage.

The curves displayed in Figure 5.32 are generated by computing the relative solution time

$$relative\ Solution\ time(X, Y, I) = \frac{T_{\text{sol,X,I}} - T_{\text{sol,Y,I}}}{\min(T_{\text{sol,X,I}}, T_{\text{sol,Y,I}})}$$

where $T_{\text{sol,X,I}}$ denotes the solution time of solver X for instance I. If the curve gives a negative value, then the solution time of the comparator is negative. If the curve has a value near zero, then both solver settings lead to a similar solution time for the specific instance. If the solution time is positive, then the solution time of the implementation is larger than the solution time of the comparator.

Figure 5.32 shows that the column generation approach is outperformed by GUROBI solving the presolved break-based formulation with up to 28 threads. One can additionally see that the column generation approach has a similar performance as GUROBI using up to 8 threads. Often, the column generation approach outperforms GUROBI by using up to 28 threads and solving the non-presolved formulation. Thus, the column generation approach will be strong in the cases when the presolving is not able to delete large sets of variables. Then, the advantage of column generation will appear, and the branch-and-bound nodes can be solved faster than in the case of the complete problem formulation.

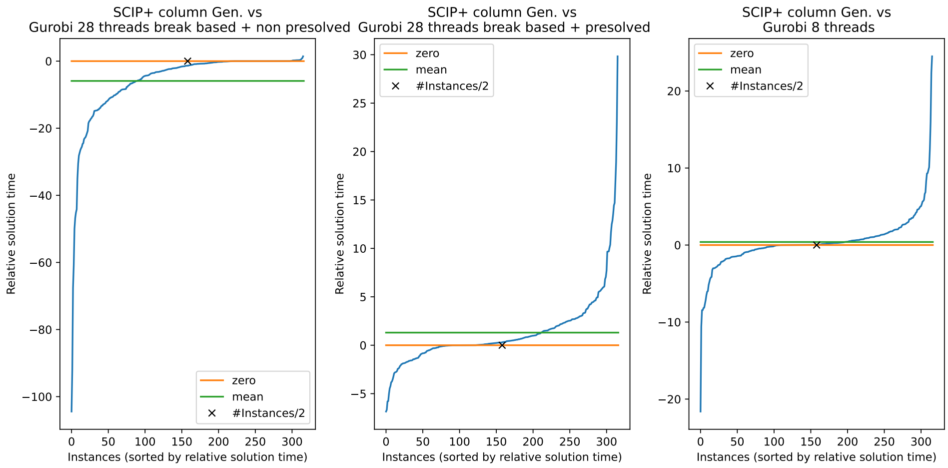


Figure 5.32: Visualization of the number of faster-solved instances by the column generation approach in comparison to different GUROBI settings. The blue line shows the curve of the relative solution time of the two selected solvers. The green line shows the mean value of the relative solution time. The orange line shows the x-axis and the x denotes the half of the number of instances. If two solvers perform always equally well, then the relative solution time is equal to the zero line. The plot in the first column shows the column generation approach in comparison to GUROBI using 28 threads and the non-presolved break-based formulation. The column in the middle shows the column generation approach and GUROBI using up to 28 threads and the presolved break-based formulation. The last column shows the column generation approach and GUROBIusing 8 threads to solve the presolved formulation.

Table 5.6: Comparison between the break-based formulation without presolving and the column generation approach, considering the number of variables. This involves examining both the total number of variables in the initial formulation and the percentage of variables remaining after presolving or when utilizing the column generation method after finishing the complete solution process (with a time limit of 3600s).

instance	initial	presolved (%)	Col. Gen (%)	Col. gen initial (%)	priced (%)
laXX_s_s	15079	81.8	36.4	25.7	10.8
laXX_s_m	24358	81.8	33.6	19.5	14.2
laXX_s_l	35442	84.5	30.8	16.0	14.8
laXX_s_h	47296	88.4	27.8	13.8	14.0
laXX_m_s	12111	73.0	35.6	29.1	6.49
laXX_m_m	21390	71.6	28.1	20.5	7.6
laXX_m_l	32474	72.7	25.3	16.3	9.02
laXX_m_h	44328	77.3	22.9	13.9	8.99
laXX_l_s	9372	44.9	39.0	34.4	4.67
laXX_l_m	18645	63.5	26.5	21.9	4.58
laXX_l_l	29713	64.2	22.2	16.8	5.35
laXX_l_h	41557	67.0	19.8	14.1	5.67
laXX_r_s	12166	69.7	36.8	28.4	8.35
laXX_r_m	21400	73.9	29.7	20.4	9.34
laXX_r_l	32131	73.6	25.0	16.2	8.78
laXX_r_h	44418	77.7	22.9	13.8	9.14

Table 5.6 shows the averaged number of variables per instance size (initial variables) and the number of variables after only MILP-solver presolving (presolved). Additionally, the average number variables are used within the column generation approach (CC max vars). These variables are divided into sets of initial variables (initial vars) and priced variables (priced breaks). Table 5.6 shows that the column generation approach can lead to significantly smaller problems. The number of priced variables is always smaller than 15% of the total number of the original formulation, and the number of initial variables dominates the number of variables. However, there are also instances where the number of priced variables exceeds the number of initial variables. These are the instances with a large number of break variables.

The branch-and-bound algorithm outperforms the branch-and-price algorithm. The strong presolving algorithm is the reason for this. Table 5.7 shows the number of variables after applying the presolving (initial variables) and the percentage of variables after MILP-presolving (presolved).

To show that a presolving algorithm is a crucial tool, Table 5.7 shows the number of variables after finishing the column generation approach (CC max vars), the percentage of initial variables after presolving (remaining vars), the percentage of remaining variables after MILP presolving (presolved), the percentage of priced variables and the allocation key of initial variables (initial vars) and priced variables (priced breaks).

Table 5.7: Comparison between the break-based formulation with our presolving and the column generation approach, considering the number of variables. This involves examining both the total number of variables in the initial formulation and the percentage of variables remaining after presolving or when utilizing the column generation method.

instance	initial	presolved (%)	Col. Gen (%)	Col. gen initial (%)	priced (%)
laXX_s_s	6524	94.1	84.2	59.3	24.9
laXX_s_m	11503	92.5	71.2	41.2	30.0
laXX_s_l	17766	92.6	61.4	31.8	29.5
laXX_s_h	24714	92.6	53.3	26.4	26.9
laXX_m_s	3834	97.7	112.0	91.8	20.5
laXX_m_m	5781	96.5	104.0	75.9	28.1
laXX_m_l	9012	95.1	91.3	58.8	32.5
laXX_m_h	13349	94.5	76.1	46.2	29.9
laXX_l_s	3327	97.1	110.0	96.8	13.2
laXX_l_m	4414	97.6	112.0	92.6	19.4
laXX_l_l	5998	96.9	110.0	83.4	26.5
laXX_l_h	7948	96.4	104.0	73.8	29.7
laXX_r_s	4517	95.6	99.0	76.5	22.5
laXX_r_m	6974	94.5	91.1	62.5	28.7
laXX_r_l	10220	94.4	78.5	50.9	27.6
laXX_r_h	15511	93.8	65.7	39.5	26.2

Table 5.7 shows that the presolved formulation often uses fewer or a similar number of variables than the column generation approach. The number of variables of the column generation approach can be larger since we do not apply each presolving rule within the pricing algorithm. The presolving rule 4.1.41 is not applied in column generation since this rule is too time-consuming. However, if the optimal primal solution is computed within the early stages of the solution process, many of these variables are not created because their reduced costs are too expensive. Since the branch-and-bound algorithm considers a problem with a similar number of variables and can generate stronger valid inequalities, it is finally faster.

5.4.5 Summary and Confirmation of Performance

This section summarizes the main results of this work and shows that the approaches are purposeful. Firstly, we give an overview of multiple runs to compare the implementation and one of the currently fastest MILP solvers (Gurobi). Figure 5.33 shows the perfor-

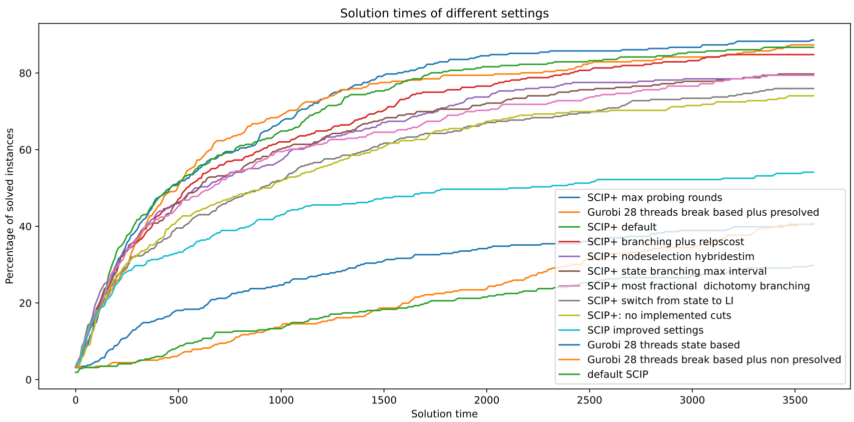


Figure 5.33: This figure shows the number of solved instances over the bounded time window. The figure includes the number of solved instances by the default Scip implementation (default Scip), Gurobi using up to 8 threads (grb eight threads), Gurobi using up to 8 threads to solve the initial model (grb eight threads: initial model), Gurobi using up to 28 threads (grb 28 threads), Gurobi using up to 8 threads to solve the straight forward formulation (grb eight threads: straightforward model), and our implementation.

mance of the different solvers, models and implementations. An implementation A is better as a different implementation B if the resulting performance curves gradient, corresponding to implementation A, is steeper than the curve, corresponding to B. The best and also an unrealistic case is that all instances are solved after 0 percent of the time window. Then, the implementation can solve more instances within a shorter time window. This

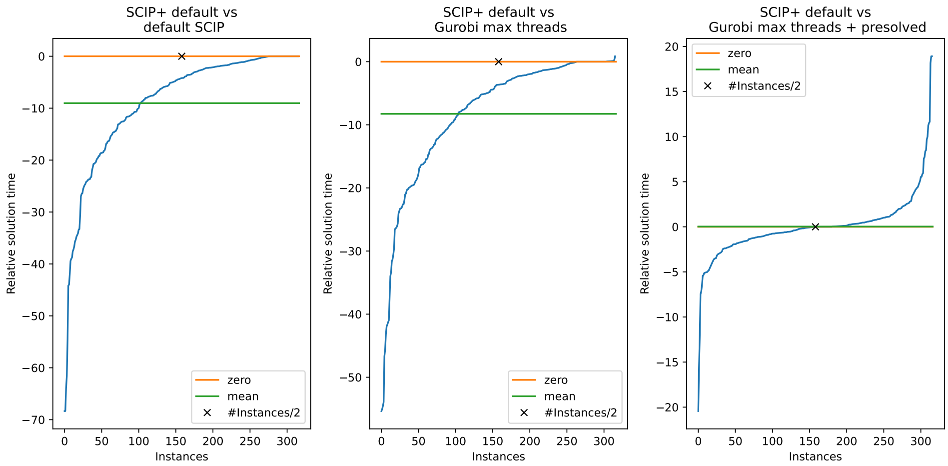


Figure 5.34: Performance comparisons of SCIP+ in comparison to SCIP in default settings using the presolved break-based formulation, and to GUROBI using 28 thread and the non-presolved break-based formulation, and to GUROBI using 28 threads and the presolved break-based formulation.

figure shows impressively that GUROBI, using up to 28 threads solving the presolved break-based formulation, outperforms all other implementations. However, one can see that the state-based formulation can only solve 40% of the generated instances. Analogously, the complete initial model solved by GUROBI also only achieves 40% within the chosen time window. If we initially reduce the number of breaks, the performance of GUROBI using up to 8 or 28 threads is as good as an average performance score of the implementation SCIP+. One can see that most of the implementations solve about 70% of all instances. This result is similar to GUROBI using 28 threads and the presolved formulation. If we disable the central parts of the solution algorithm, the performance decreases drastically. The following Figure 5.34 visualizes whether the implementation and formulation are outperforming another solver or implementation. If the curve gives a negative value, then the solution time of the comparator is negative. If the curve has a value near zero, then both solver settings lead to a similar solution time for the specific instance. If the solution time is positive, then the solution time of the implementation is larger than the solution time of the comparator. Figure 5.34 shows three pictures. In each of these pictures, the solution time of the implementation is compared to one other solver, either GUROBI using 28 threads and the presolved formulation, GUROBI using 28 threads and the non-presolved formulation, and SCIP using its default settings. One can see in each of these pictures that the devised branch-and-bound algorithm is as good as GUROBI using 28 threads on the presolved formulation. The difference of the relative solution time becomes more significant if we compare SCIP+ and SCIP. The improvement of the performance becomes more significant if we consider the state-based formulation and the non-presolved formulation solved by GUROBI using 28 threads. Figure 5.35 shows the performance improvement of the implemented branch-and-bound algorithm in comparison to the state-based formulation and the commercial solver, using (parallel) variable branching. There is almost no instance that is solved faster by GUROBI with up to 28 threads and the state-based formulation than by the branch-and-bound algorithm. In addition, if GUROBI with 28 threads solves the non-presolved formulation, then the branch-and-bound algorithm outperforms GUROBI. Note that the usage of the presolved formulation led to a similar performance of GUROBI using 28 threads and the implementation of SCIP+.

Implementing branch-and-bound algorithms has led to a significant improvement in performance. The provided algorithm is able to solve a similar number of instances within the same time limit using fewer threads. It is important to mention that the implementation in SCIP leads to solution times that are comparable to GUROBI with 28 threads on the evaluated test instances. The implementation is exclusively single-threaded, within GUROBI is high-grade parallel. The usage of parallel programming in SCIP and or the implementation of the branching rules in commercial solvers would lead again to a more significant increase in performance.

We started with a formulation that is not able to consider negative energy prices and also requires a lot of time to solve the major part of our created test instances. We developed a new problem formulation and discovered presolving rules leading to a compact

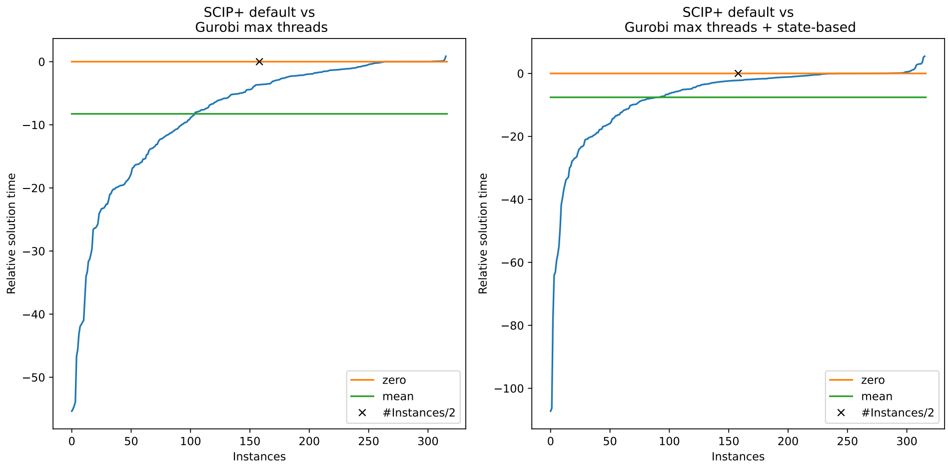


Figure 5.35: Performance comparisons of Scip+ in comparison to Gurobi using 28 threads and the non-presolved break-based formulation, and to Gurobi using 28 threads solving the state-based formulation.

model, such that 80% of each of our test instances is solved within 3600 seconds. The solution process requires a lot of branch-and-bound nodes, which leads us to the implemented branching rules, which reduce the required number of nodes for successfully solving most of the instances. The consideration of further inequalities and column generation provides further possibilities of how the problems can be solved. Since near-optimal solutions are often good enough to be used in reality, different heuristic approaches are presented. Thus, this thesis provides multiple approaches to solve the job-shop scheduling problem with flexible energy prices and time windows to optimality or only heuristically, and the solution times are as fast as the solution times of one of the fastest commercial MILP solvers, which is tuned to solve the problem highly parallel, while our code is only able to run single threaded.

Chapter 6

Conclusions

This dissertation considers the job-shop scheduling problem with flexible energy prices and time windows. We focused on devising, implementing, and documenting a branch-and-bound approach to solve this complex combinatorial optimization problem efficiently.

To address the problem-specific challenges, we proposed a partial Dantzig–Wolfe reformulation to explicitly describe ramping and offline periods. Introducing the variables allows the consideration of non-linear energy demands and accurately mirrors the actual manufacturing conditions. This formulation was called the break-based formulation. Initially, all instances were equally hard to solve, regardless of whether we used the state-based or fraction-based formulation, with only marginal differences in the average solution time. However, we proved that the break-based formulation provides a more precise description of the integral feasible solutions to the job-shop scheduling problem with flexible energy prices and time windows than the state-based formulation.

We have implemented a column generation algorithm for the break variables, which allows us to keep the number of break variables small. An additional way to solve the problem could be the implementation of a variable pricer for groups of processing starts of tasks from the same job and to build another problem formulation with different advantages and disadvantages.

Throughout our research, we explored the challenges arising when solving the formulated integer program by commercial solvers since the time-indexed formulation suffers from unbalanced branches. Therefore, we devised and implemented different constraint-based branching rules. These branchings are applied to overcome the problems of unbalanced branches, which often occur in set-packing problems. The different branching rules were embedded in a branching algorithm, where a logic decides which branching rule is the most suitable one. The experimental results show that our logic, which chooses the most suitable branching rule, can still be improved by further constrained branching rules and rules to switch on classical variable branching. The fractionality of the workload and the spreading of task processing within the time window are two issues that our branching rules address. We identified that the computation of the optimal execution order is only a secondary concern. The objective costs are primarily determined by the workload, and therefore, our branching mainly tries to apply the workload branching rule. However, we have also observed that for nearly integral workloads, our branching approach is no longer the best decision. At this stage, further research must explore whether we should prioritize workload branching or time window arrangement on machines based on the fractional solution. We have taken initial steps, but the experimental results have revealed significant potential for improvement. Furthermore, it has been demonstrated that classical variable branching presents itself as a conceivable alternative when the machine profile is nearly integral. These aspects of branching and branching rule selection should also be further investigated.

Our problem-specific presolving rules are capable of efficiently and significantly reducing the size of the problem formulation. This reduction of the problem size leads to a speedup when only solving the model with black-box solvers. This thesis examines various subproblems within the job-shop scheduling problem with flexible energy prices and time windows and provides combinatorial conditions or algorithms for solving these problems: for example, the double knapsack substructure and the presolving by set dominated columns. It turns out that some rules are only applicable as presolving rules. Although the task variables represent a large part of the problem variables, we completely disregard them in our presolving rules. This explains why additional presolving rules for the task variables must be implemented to reduce the problem size further and improve the lower bound on the best objective value. In the current algorithm, the task variables are only reduced through reduced-cost propagation and branching. However, it should be possible, based on the objective, to recognize that certain variable configurations are not feasible in optimal solutions. We provide valid constraints for the polytope of the break-based formulation. There are clique-cuts, GUB cover constraints and constraints from linear ordering. In this thesis, GUB cover constraints were considered. In the scheduling literature, there is a detailed examination of GUB cover constraints with a right-hand side of 1 and 2 in the case of single machine scheduling. However, we only provide a lifting scheme for the break variables in the case of right-hand-side 1. In the other case, a lifting rule for breaks can also be derived. We also derived valid constraints from the linear ordering problem. These inequalities, derived from the subproblem of the linear ordering problems for tasks on the same machine, consider only the most essential inequalities of the linear ordering problems. Through a Benders decomposition, which uses the linear ordering problem with additional inequalities as a subproblem, we might be able to separate even stronger inequalities. Again, taking the step of lifting break variables into the separated inequalities would be highly beneficial. Another point would be the improvement of the model. We have already attempted to enhance the basic model through knapsack inequalities. It has been found that there are knapsack inequalities that link the ramping of the machines through a job chain. Deriving additional types of such inequalities could lead to an improved formulation. Most of our approaches aim to strengthen the dual bound. To compute primal solutions, we implemented a diverse set of heuristics suitable for various problem stages. We incorporated list-scheduling heuristics and list-scheduling-based neighborhood searches, which initially explore the solution space to find an initial primal solution. Diving heuristics have been implemented, attempting infrequently in the branch-and-bound tree to find an improving primal solution in the depths of the unexplored branch-and-bound tree by fixing only the task variables. Additionally, we developed a dynamic program for improving the current solution through a local search. In addition, we incorporated a genetic algorithm into the implementation, which can initially provide a good primal solution. By combining different primal algorithms with a branching rule focused on the dual bound, presolving and propagation rules, and cutting planes, a solid solver has been developed for the job-shop scheduling problem with flexible energy prices and time windows that can solve multiple instances faster than commercial (untrained but highly parallel) solvers. In order to apply the implemented techniques in realistic cases, a heuristic application of our knowledge of the branch-and-bound algorithm is required. A depth-first search, using assignment-constraint branching to explore a path in depth, can quickly and efficiently find near-optimal solutions if we can estimate well which path in the branch-and-bound tree leads us in the right direction. Some of those heuristic approaches would speed up our solution approach since we would be able to compute near-optimal solutions initially. Additionally, such approaches would also be of economic interest, as they could efficiently compute suitable solutions for large instances in a relatively short time, while the computation of the optimal solution requires too much time. Furthermore, an attempt should be made to implement the well-known shifting bottleneck heuristic for the job-shop scheduling problem with energy prices by devising an approximate objective function and neglecting breaks and standby. This could lead to the early computation of additional good solutions for the problem. Considering further problem variants, such as allowing the violation of precedence relationships with additional penalty costs or incorporating stochastic elements into the objective function, such as the actual energy consumption of machines or an uncertain energy price, could be other use cases that are of interest.

In conclusion, our study on the job-shop scheduling problem with flexible energy prices and time windows has led to the development of a competitive branch-and-bound algorithm and branch-and-price algorithm. This work has demonstrated that considering energy prices while solving the scheduling problem to optimality is possible in a realistic period of time in the case of small instances. By employing a time-indexed model, this approach can be easily extended to accommodate various complex conditions, including resource constraints, requirements for simultaneous machine ramp-ups and ramp-downs, or energy consumption spikes. Thus, a tool now exists through which optimization can be conducted, or at the very least, a practical solution can be computed in an acceptable time.

Appendix A

Appendix

A.1 Settings of Our Implementation

Table A.1: Important setting changes of our implementation.

Setting	Value	Setting	Value
limits/gap	1e-06	presolving/maxrestarts	0
presolving/donotmultaggr	TRUE	presolving/donotaggr	TRUE
separating/maxlocalbounddist	1	separating/maxstallroundsroot	-1
separating/poolfreq	1	constraints/knapsack/sepafreq	1
constraints/setppc/sepafreq	1	presolving/domcol/numminpairs	1048576
presolving/dualagg/maxrounds	-1	nodeselection/bfs/stdpriority	max
heuristics/adaptivediving/freq	-1	heuristics/conflictdiving/freq	-1
heuristics/distributiondiving/freq	-1	heuristics/farkasdiving/freq	-1
heuristics/fracdiving/freq	-1	heuristics/guideddiving/freq	-1
heuristics/linesearchdiving/freq	-1	heuristics/lpface/freq	-1
heuristics/alns/freq	10	heuristics/nlpdiving/freq	-1
heuristics/objpscostdiving/freq	-1	heuristics/pscostdiving/freq	-1
heuristics/rootsoldiving/freq	-1	heuristics/veclendiving/freq	-1
propagating/dualfix/freq	1	separating/clique/freq	1
separating/clique/maxbounddist	1	separating/aggregation/freq	-1
separating/gomory/freq	-1	separating/impliedbounds/freq	1
separating/zerohalf/freq	1	separating/zerohalf/maxbounddist	0
separating/zerohalf/maxslack	1	separating/zerohalf/maxslackroot	1
separating/zerohalf/badscore	0	MIP/addCliquesConflict	1
MIP/addPrecedenceAsClique	1	MIP/aggregatePrec	1
MIP/ownPrec	1	branching/mscg_ub/rule_Time	W
branching/mscg_status/priority	max	branching/mscg_status/maxdepth	100
propagating/mscg_prob/obj	1	propagating/mscg_prob/len	1
propagating/mscg_prob/binpack	1	propagating/mscg_prob/overlap	1
propagating/mscg_prob/unn	1	propagating/mscg_prob/breaks	1
heuristics/DivM/lpsolvefreq	1	separating/wolsey_rhs2/freq	1
heuristics/DivM/maxdiveavgquot	10	separating/VDAkker1_rhs1/freq	1
heuristics/DL/neighbourhood	2	heuristics/DL/freq	10
heuristics/TS/freq	-1	heuristics/heurFirstIn/pe	0.1
heuristics/heurFirstIn/pm	0.3	heuristics/heurFirstIn/size_pop	2
heuristics/RM/freq	-1	heuristics/LB/freq	5
heuristics/DivE/freq	1	heuristics/DivE/freqofs	0
heuristics/DivE/maxdepth	10	heuristics/DivE/maxdiveubquot	1e-05
heuristics/DivE/maxdiveavgquot	100	heuristics/DivE/lpsolvefreq	1
heuristics/DivM/freq	1	heuristics/DivM/freqofs	1
heuristics/DivM/maxdepth	10	heuristics/DivM/maxdiveubquot	1e-05

A.2 Instances

A.2.1 Dataorig_ver_1 with $T = 72$

Table A.2: Job Shop Scheduling Data

Job	Operation	Machine	Setup Time	Processing Time
0	0	0	3	4
0	1	1	3	4
0	2	3	1	6
0	3	4	1	6
0	4	1	4	4
1	0	2	3	4
1	1	1	3	4
1	2	4	1	5
1	3	3	1	5
1	4	0	3	4
2	0	0	4	5
2	1	1	4	5
2	2	2	4	8
2	3	4	3	4
3	0	2	2	5
3	1	1	2	5
3	2	3	1	4
3	3	4	1	4
4	0	0	2	3
4	1	1	2	3
4	2	2	2	3

Table A.3: Machine Information

Parameter	$m = 0$	$m = 1$	$m = 2$	$m = 3$	$m = 4$
Ramping up duration	3	3	3	2	1
Ramping down duration	2	2	2	1	1
D_m^{off}	0	0	0	0	0
D_m^{ru}	18	10	5	4	2
D_m^{st}	8	8	8	3	3
D_m^{se}	20	20	20	6	6
D_m^{pr}	7	1	0.5	0.5	0.5
D_m^{rd}	5	5	5	2	2

Table A.4: Time windows of the Jobs

Job id	start time	due date
0	0	72
1	8	72
2	16	72
3	24	72
4	48	72

A.2.2 la01_7_s_s with $T = 108$

Table A.5: Operation Section Data

Job	Operation	Machine	Setup Time	Processing Time
0	0	1	1	2
0	1	0	2	4
0	2	4	4	8
0	3	3	2	4
0	4	2	2	4
1	0	0	1	2
1	1	3	2	4
1	2	4	1	2
1	3	2	1	2
1	4	1	3	6
2	0	3	2	4
2	1	4	4	8
2	2	1	2	4
2	3	2	2	4
2	4	0	1	2
3	0	1	3	6
3	1	0	2	4
3	2	4	3	6
3	3	2	3	6
3	4	3	3	6
4	0	0	3	6
4	1	3	2	4
4	2	2	3	6
4	3	1	1	2
4	4	4	2	4
5	0	1	2	4
5	1	2	2	4
5	2	4	3	6
5	3	0	4	8
5	4	3	3	6
6	0	3	3	6
6	1	4	3	6
6	2	1	3	6
6	3	2	3	6
6	4	0	4	8
7	0	2	2	4
7	1	0	2	4
7	2	1	2	4
7	3	3	1	2
7	4	4	3	6
8	0	3	1	2
8	1	1	2	4
8	2	4	1	2
8	3	0	2	4
8	4	2	4	8
9	0	4	3	6
9	1	3	3	6
9	2	2	2	4
9	3	1	3	6
9	4	0	4	8

Table A.6: Machine Information

Parameter	$m=0$	$m=1$	$m=2$	$m=3$	$m=4$
Ramping up duration	4	4	4	4	4
Ramping down duration	2	2	2	2	2
D_m^{off}	0	0	0	0	0
D_m^{ru}	4	1	5	1	6
D_m^{st}	3	4	1	1	4
D_m^{se}	7	4	6	7	4
D_m^{pr}	2	1	1	1	3
D_m^{rd}	7	4	4	4	5

Table A.7: Time windows of the Jobs

Job id	start time	due date
0	9	108
1	9	108
2	9	108
3	9	108
4	9	108
5	9	108
6	9	108
7	9	108
8	9	108
9	9	108

A.2.3 la02_8_r_s with $T = 99$

Table A.8: Operation Section Data

Job	Operation	Machine	Setup Time	Processing Time
0	0	0	1	2
0	1	3	3	6
0	2	1	2	4
0	3	4	3	6
0	4	2	1	2
1	0	4	1	2
1	1	2	2	4
1	2	0	1	2
1	3	1	1	2
1	4	3	3	6
2	0	1	3	6
2	1	2	1	2
2	2	4	1	2
2	3	0	2	4
2	4	3	4	8
3	0	2	3	6
3	1	1	3	6
3	2	4	4	8
3	3	0	2	4
3	4	3	3	6
4	0	4	1	2
4	1	0	2	4
4	2	3	2	4
4	3	2	1	2
4	4	1	2	4
5	0	1	3	6
5	1	0	4	8
5	2	4	2	4
5	3	3	1	2
5	4	2	3	6
6	0	4	1	2
6	1	1	1	2
6	2	3	1	2
6	3	0	3	6
6	4	2	2	4
7	0	1	3	6
7	1	0	4	8
7	2	2	4	8
7	3	3	2	4
7	4	4	3	6
8	0	4	1	2
8	1	0	3	6
8	2	2	2	4
8	3	1	2	4
8	4	3	2	4
9	0	4	3	6
9	1	2	1	2
9	2	1	2	4
9	3	3	3	6
9	4	0	3	6

Table A.9: Machine Information

Parameter	$m=0$	$m=1$	$m=2$	$m=3$	$m=4$
Ramping up duration	12	4	14	2	11
Ramping down duration	9	6	8	1	2
D_m^{off}	0	0	0	0	0
D_m^{ru}	2	2	6	1	28
D_m^{st}	3	6	5	5	5
D_m^{se}	7	10	15	10	21
D_m^{pr}	3	3	3	3	3
D_m^{rd}	20	8	14	14	35

Table A.10: Time windows of the Jobs

Job id	start time	due date
0	0	99
1	0	99
2	0	99
3	0	99
4	0	99
5	0	99
6	0	99
7	0	99
8	0	99
9	0	99

Details of the Experimental Results

instance	vars	cons	nodes	time	relative root	relative primal	gap
la01_0_l_h	dual =	217100.0	opt =	219400.0	gap	= 0.01	
State-Based	12259	22357	10155	3600.03	0.02842	0.003683	0.01739
Break based	38275	12042	2001	3600.12	0.01026	0.004932	0.01481
SCIP+	10231	860	2925	3600.17	0.008532	0.004203	0.008165
SCIP+: col. gen.	9355	995	2629	1396.19	0.001706	0.0	0.0
Presolved break based	14749	12091	42679	773.54	0.01	0.0	0.0
la01_0_l_l	dual =	242000.0	opt =	244300.0	gap	= 0.0095	
State-Based	10282	19171	15132	3600.04	0.04303	0.0004912	0.01928
Break based	25105	10164	22733	3600.31	0.009647	0.0	0.002799
SCIP+	7430	770	3146	1308.53	0.008557	0.0	0.0
SCIP+: col. gen.	8177	905	1731	977.68	0.001292	0.0	0.0
Presolved break based	8862	10210	6481	421.98	0.009	0.0	0.0
la01_0_l_m	dual =	266200.0	opt =	267000.0	gap	= 0.0031	
State-Based	8302	15931	27309	3600.06	0.04156	0.003355	0.01847
Break based	16042	8292	4074	1018.24	0.003206	0.0	0.0
SCIP+	4952	680	103	36.08	0.002666	0.0	0.0
SCIP+: col. gen.	5328	815	409	95.82	0.0006223	0.0	0.0
Presolved break based	6243	8343	969	85.89	0.0031	0.0	0.0
la01_0_l_s	dual =	277200.0	opt =	277700.0	gap	= 0.0018	
State-Based	6318	12636	39086	3600.12	0.02236	0.0	0.004961
Break based	8829	6425	4164	109.72	0.001809	0.0	0.0
SCIP+	3723	590	522	70.31	0.001795	0.0	0.0
SCIP+: col. gen.	4081	725	1162	100.9	0.0007086	0.0	0.0
Presolved break based	3822	6454	1100	30.37	0.0017	0.0	9.4e-05
la01_0_m_h	dual =	131300.0	opt =	132200.0	gap	= 0.0072	
State-Based	12859	18967	13385	3600.03	0.02291	0.004159	0.01412
Break based	44761	12674	6377	1683.32	0.00683	0.0	0.0
SCIP+	17380	890	852	564.06	0.005792	0.0	0.0
SCIP+: col. gen.	10742	965	1495	889.79	0.001497	0.0	0.0
Presolved break based	23171	12717	4159	263.9	0.007	0.0	0.0
la01_0_m_l	dual =	146700.0	opt =	147300.0	gap	= 0.0041	
State-Based	10882	16303	12234	2873.79	0.02902	0.0	0.0
Break based	30226	10788	799	1232.96	0.004267	0.0	0.0
SCIP+	11435	800	208	209.54	0.00366	0.0	0.0
SCIP+: col. gen.	10322	875	711	556.6	0.0008354	0.0	0.0
Presolved break based	14465	10837	761	202.15	0.0042	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la01_0_m_m	dual =	163600.0	opt =	164800.0	gap	= 0.0074	
State-Based	8899	13567	14674	3600.05	0.04267	0.0	0.02454
Break based	20086	8922	10756	3583.2	0.007461	0.0	0.0
SCIP+	7131	710	740	377.94	0.00688	0.0	0.0
SCIP+: col. gen.	6855	785	819	444.63	0.0004147	0.0	0.0
Presolved break based	8975	8965	9966	435.36	0.0075	0.0	0.0
la01_0_m_s	dual =	173500.0	opt =	174900.0	gap	= 0.0081	
State-Based	6919	10867	27090	3600.05	0.03912	0.001601	0.01471
Break based	11694	7042	53474	3600.67	0.008072	0.0	0.002959
SCIP+	4345	620	12599	3223.92	0.004663	0.0	0.0
SCIP+: col. gen.	5022	695	12760	2494.52	0.0002011	0.0	0.0
Presolved break based	5308	7085	59986	613.81	0.0081	0.0	0.0
la01_0_r_h	dual =	107100.0	opt =	107800.0	gap	= 0.0069	
State-Based	12625	19943	19743	3600.03	0.0195	0.0	0.003975
Break based	43032	12409	13609	2320.99	0.007342	0.0	0.0
SCIP+	19272	881	2421	1392.0	0.0	0.0	0.0
SCIP+: col. gen.	11038	974	2029	975.55	0.001493	0.0	0.0
Presolved break based	22251	12456	4156	500.6	0.0072	0.0	0.0
la01_0_r_l	dual =	125700.0	opt =	127400.0	gap	= 0.013	
State-Based	10602	17039	9129	3600.05	0.06804	0.001924	0.0283
Break based	29187	10481	1826	3600.15	0.01279	0.0	0.009688
SCIP+	10270	791	4313	3600.12	0.01091	0.0	0.006318
SCIP+: col. gen.	11514	884	7481	3600.04	0.0003124	0.0	0.002866
Presolved break based	12275	10520	81818	3600.26	0.013	0.0	0.002
la01_0_r_m	dual =	187800.0	opt =	188900.0	gap	= 0.006	
State-Based	8646	13826	29689	3614.51	0.03723	0.0	0.004573
Break based	19544	8643	22151	2190.34	0.005954	0.0	0.0
SCIP+	7022	704	2873	695.12	0.005047	0.0	0.0
SCIP+: col. gen.	6746	791	3358	535.6	0.001138	0.0	0.0
Presolved break based	9181	8684	7369	292.62	0.0058	0.0	0.0
la01_0_r_s	dual =	112800.0	opt =	113100.0	gap	= 0.003	
State-Based	7057	9053	58037	3614.37	0.04715	0.0	0.002463
Break based	14032	7095	11407	886.87	0.003193	0.0	0.0
SCIP+	8652	640	4517	1212.56	0.002558	0.0	0.0
SCIP+: col. gen.	6559	675	2280	474.65	0.0004512	0.0	0.0
Presolved break based	9881	7136	6146	284.21	0.0031	0.0	0.0
la01_0_s_h	dual =	53010.0	opt =	53920.0	gap	= 0.017	
State-Based	13561	14562	22160	3600.09	0.03036	0.001261	0.02194
Break based	52818	13409	1645	3600.56	0.01693	0.001669	0.0163
SCIP+	19717	925	3738	3600.35	0.01463	0.0001298	0.007902
SCIP+: col. gen.	15204	930	8123	3600.04	0.002262	0.00102	0.008213
Presolved break based	24503	13457	115282	2281.8	0.018	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la01_0_s_l	dual =	59760.0	opt =	60490.0	gap	= 0.012	
State-Based	11579	12482	25511	3600.02	0.02481	0.005637	0.01929
Break based	39357	11519	10171	3600.22	0.01205	0.0	0.00773
SCIP+	14717	835	4729	3253.97	0.009868	0.0	0.0
SCIP+: col. gen.	14270	840	6229	3511.01	0.00127	0.0	0.0
Presolved break based	18643	11562	52481	1236.74	0.012	0.0	0.0
la01_0_s_m	dual =	67160.0	opt =	67950.0	gap	= 0.012	
State-Based	9599	10412	29294	3600.09	0.0397	0.000986	0.01383
Break based	25484	9648	2140	3600.19	0.01165	0.007373	0.01775
SCIP+	9774	745	7705	3600.1	0.01145	0.0003532	0.005152
SCIP+: col. gen.	11137	750	8408	3600.03	0.001828	0.0003532	0.004343
Presolved break based	11231	9695	17617	639.68	0.012	0.0	0.0
la01_0_s_s	dual =	72640.0	opt =	73190.0	gap	= 0.0075	
State-Based	7619	8342	41121	3600.1	0.03173	0.0	0.006449
Break based	15692	7767	3374	970.63	0.007545	0.0	0.0
SCIP+	5590	655	324	98.32	0.00589	0.0	0.0
SCIP+: col. gen.	5750	660	526	151.5	0.001034	0.0	0.0
Presolved break based	6423	7820	10125	224.43	0.0073	0.0	0.0
la01_1_l_h	dual =	6381.0	opt =	6381.0	gap	= 0.0	
State-Based	12261	22342	3336	2832.72	0.0	0.0	0.0
Break based	38042	12042	1	2476.35	0.0	0.0	0.0
SCIP+	8816	860	193	2003.46	0.0	0.0	0.0
SCIP+: col. gen.	8150	995	92	164.42	0.0	0.0	0.0
Presolved break based	14778	12090	1	863.73	0.0	0.0	0.0
la01_1_l_l	dual =	6381.0	opt =	6381.0	gap	= 0.0	
State-Based	10281	19102	3512	1055.38	0.0	0.0	0.0
Break based	24862	10164	1	2303.89	0.0	0.0	0.0
SCIP+	6222	770	48	476.01	0.0	0.0	0.0
SCIP+: col. gen.	7858	905	1309	2552.23	0.0	0.0	0.0
Presolved break based	7541	10210	173	690.49	0.0	0.0	0.0
la01_1_l_m	dual =	6381.0	opt =	6381.0	gap	= -0.0	
State-Based	8301	15862	3974	1648.88	0.0	0.0	0.0
Break based	15799	8292	1	436.74	0.0	0.0	0.0
SCIP+	4990	680	144	397.48	0.0	0.0	0.0
SCIP+: col. gen.	5830	815	972	836.38	0.0	0.0	0.0
Presolved break based	5090	8331	1	270.62	0.0	0.0	0.0
la01_1_l_s	dual =	6381.0	opt =	6381.0	gap	= -0.0	
State-Based	6320	12621	1	122.82	0.0	0.0	0.0
Break based	8385	6403	1	169.82	0.0	0.0	0.0
SCIP+	3819	590	81	142.51	0.0	0.0	0.0
SCIP+: col. gen.	4279	725	607	414.94	0.0	0.0	0.0
Presolved break based	3820	6446	1	22.54	0.0	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la01_1_m_h	dual =	4067.0	opt =	4067.0	gap	= -0.0	
State-Based	12859	18957	3713	2559.12	0.0	0.0	0.0
Break based	44590	12674	1	3600.13	0.0	0.2009	0.2009
SCIP+	18340	890	532	3600.17	0.0	0.001967	0.001967
SCIP+: col. gen.	9728	965	334	843.31	0.0	0.0	0.0
Presolved break based	25993	12722	1	78.17	0.0	0.0	0.0
la01_1_m_l	dual =	4067.0	opt =	4067.0	gap	= 0.0	
State-Based	10879	16257	2460	1375.81	0.0	0.0	0.0
Break based	30043	10788	1	3600.1	0.0	0.04623	0.04623
SCIP+	10551	800	228	742.99	0.0	0.0	0.0
SCIP+: col. gen.	7804	875	217	305.78	0.0	0.0	0.0
Presolved break based	14289	10838	1	252.34	0.0	0.0	0.0
la01_1_m_m	dual =	4067.0	opt =	4067.0	gap	= 0.0	
State-Based	8899	13557	2270	644.45	0.0	0.0	0.0
Break based	19906	8922	1	3600.11	0.0	0.001229	0.001229
SCIP+	6127	710	38	171.97	0.0	0.0	0.0
SCIP+: col. gen.	5988	785	123	125.87	0.0	0.0	0.0
Presolved break based	7928	8970	1	324.84	0.0	0.0	0.0
la01_1_m_s	dual =	4067.0	opt =	4067.0	gap	= -0.0	
State-Based	6919	10857	1	136.26	0.0	0.0	0.0
Break based	11391	7033	1	810.35	0.0	0.0	0.0
SCIP+	4210	620	196	408.29	0.0	0.0	0.0
SCIP+: col. gen.	4626	695	34	42.41	0.0	0.0	0.0
Presolved break based	4373	7075	1	24.61	0.0	0.0	0.0
la01_1_r_h	dual =	3863.0	opt =	3863.0	gap	= -0.0	
State-Based	13128	16060	1725	1653.3	0.0	0.0	0.0
Break based	49975	12920	1	3600.19	0.0	0.0005177	0.0005177
SCIP+	28550	911	50	3600.35	0.0	0.007507	0.007507
SCIP+: col. gen.	11243	944	396	1423.09	0.0	0.0	0.0
Presolved break based	33830	12955	1	519.71	0.0	0.0	0.0
la01_1_r_l	dual =	4668.0	opt =	4668.0	gap	= -0.0	
State-Based	11033	14727	2146	1372.1	0.0	0.0	0.0
Break based	35080	10902	193	3600.14	0.0	0.002785	0.002785
SCIP+	10420	813	1700	1886.59	0.0	0.0	0.0
SCIP+: col. gen.	7767	862	252	381.1	0.0	0.0	0.0
Presolved break based	13126	10941	1	346.88	0.0	0.0	0.0
la01_1_r_m	dual =	3674.0	opt =	3674.0	gap	= -0.0	
State-Based	8667	13334	1675	665.4	0.0	0.0	0.0
Break based	19663	8622	1	2217.37	0.0	0.0	0.0
SCIP+	7355	709	28	133.06	0.0	0.0	0.0
SCIP+: col. gen.	5917	786	183	199.7	0.0	0.0	0.0
Presolved break based	9987	8667	1	363.26	0.0	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la01_1_r_s	dual =	3400.0	opt =	3400.0	gap	= -0.0	
State-Based	7388	9407	2857	853.36	0.0	0.0	0.0
Break based	14080	7549	1	1487.75	0.0	0.0	0.0
SCIP+	5703	641	41	99.0	0.0	0.0	0.0
SCIP+: col. gen.	5192	674	420	324.79	0.0	0.0	0.0
Presolved break based	6482	7590	1	261.24	0.0	0.0	0.0
la01_1_s_h	dual =	1759.0	opt =	1759.0	gap	= -0.0	
State-Based	13557	14548	1456	1330.99	0.0	0.0	0.0
Break based	52717	13409	1	3601.2	0.0	0.0216	0.0216
SCIP+	37993	925	5	3600.27	1e-12	0.0216	0.0216
SCIP+: col. gen.	11517	930	384	1073.25	0.0	0.0	0.0
Presolved break based	45314	13457	1	179.84	0.0	0.0	0.0
la01_1_s_l	dual =	1759.0	opt =	1759.0	gap	= 0.0	
State-Based	11577	12478	305	473.98	0.0	0.0	0.0
Break based	39256	11519	59	3602.04	0.0	0.2871	0.2871
SCIP+	26302	835	137	2717.09	0.0	0.0	0.0
SCIP+: col. gen.	10164	840	623	1180.29	0.0	0.0	0.0
Presolved break based	32519	11567	1	544.02	0.0	0.0	0.0
la01_1_s_m	dual =	1759.0	opt =	1759.0	gap	= 0.0	
State-Based	9597	10408	4123	2535.52	0.0	0.0	0.0
Break based	25371	9648	1	2531.72	0.0	0.0	0.0
SCIP+	16231	745	623	1164.9	0.0	0.0	0.0
SCIP+: col. gen.	6482	750	84	92.93	0.0	0.0	0.0
Presolved break based	19406	9695	1	357.27	0.0	0.0	0.0
la01_1_s_s	dual =	1759.0	opt =	1759.0	gap	= 0.0	
State-Based	7617	8338	1201	327.89	0.0	0.0	0.0
Break based	15579	7767	1	1636.56	0.0	0.0	0.0
SCIP+	7780	655	127	188.38	0.0	0.0	0.0
SCIP+: col. gen.	5508	660	228	256.96	0.0	0.0	0.0
Presolved break based	10247	7817	1	275.18	0.0	0.0	0.0
la01_7_l_h	dual =	299500.0	opt =	301300.0	gap	= 0.0059	
State-Based	12259	22357	6843	3600.04	0.02301	0.001806	0.01347
Break based	38275	12042	6571	3600.57	0.005843	0.0004215	0.005333
SCIP+	11153	860	4153	3186.95	0.00456	0.0	0.0
SCIP+: col. gen.	10790	995	1805	1217.78	0.0001395	0.0	0.0
Presolved break based	15333	12086	208301	3600.25	0.0058	0.0	0.0011
la01_7_l_l	dual =	312300.0	opt =	314100.0	gap	= 0.0057	
State-Based	10279	19117	11186	3600.04	0.02318	0.003394	0.01862
Break based	25105	10164	2030	3600.39	0.005609	0.003394	0.008705
SCIP+	8015	770	2011	955.18	0.004644	0.0	0.0
SCIP+: col. gen.	8331	905	9409	3600.02	0.0002126	2.229e-05	0.00109
Presolved break based	9477	10205	160234	3226.7	0.0056	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la01_7_l_m	dual =	317100.0	opt =	318100.0	gap	= 0.0032	
State-Based	8300	15895	29808	3600.18	0.01083	0.0	0.003748
Break based	16042	8292	13858	2196.92	0.003304	0.0	0.0
SCIP+	5313	680	231	112.67	0.003017	0.0	0.0
SCIP+: col. gen.	5565	815	564	153.0	0.0002073	0.0	0.0
Presolved break based	6574	8337	17054	433.97	0.0032	0.0	0.0
la01_7_l_s	dual =	317100.0	opt =	318100.0	gap	= 0.0032	
State-Based	6318	12636	53255	3600.07	0.007618	0.0	0.000925
Break based	8742	6417	10852	530.28	0.003251	0.0	0.0
SCIP+	3794	590	203	55.18	0.002985	0.0	0.0
SCIP+: col. gen.	4171	725	411	77.42	0.0002879	0.0	0.0
Presolved break based	4163	6456	7079	185.15	0.0032	0.0	0.0
la01_7_m_h	dual =	191100.0	opt =	192800.0	gap	= 0.009	
State-Based	12859	18967	5540	3600.04	0.02622	0.01023	0.03389
Break based	44761	12674	2649	3600.21	0.008966	0.008028	0.01704
SCIP+	18542	890	3935	3600.19	0.006658	0.000726	0.004441
SCIP+: col. gen.	13551	965	5388	3600.02	0.0002249	0.000726	0.003513
Presolved break based	24520	12722	132091	2523.93	0.0089	0.0	0.0
la01_7_m_l	dual =	199600.0	opt =	200400.0	gap	= 0.0042	
State-Based	10882	16303	10388	3600.03	0.01971	0.00668	0.02074
Break based	30226	10788	2894	3600.63	0.004403	0.0237	0.02809
SCIP+	11909	800	2170	1089.54	0.003066	0.0	0.0
SCIP+: col. gen.	9458	875	8520	3600.03	0.0004197	6.984e-05	0.001167
Presolved break based	14868	10840	60649	1349.49	0.0045	0.0	0.0
la01_7_m_m	dual =	202300.0	opt =	203800.0	gap	= 0.0072	
State-Based	8899	13567	24537	3600.05	0.01468	0.0	0.008323
Break based	20089	8922	6862	3600.58	0.007377	0.004858	0.01163
SCIP+	7139	710	1887	937.88	0.006029	0.0	0.0
SCIP+: col. gen.	7155	785	3990	1511.66	0.0001151	0.0	0.0
Presolved break based	8890	8972	217244	3164.14	0.0074	0.0	8.8e-05
la01_7_m_s	dual =	202300.0	opt =	203800.0	gap	= 0.0072	
State-Based	6919	10867	36867	3600.1	0.01348	0.001345	0.006531
Break based	11697	7042	32884	3600.72	0.007079	0.0003239	0.003016
SCIP+	4442	620	1362	474.85	0.005555	0.0	0.0
SCIP+: col. gen.	5026	695	7289	1662.48	0.0001344	0.0	0.0
Presolved break based	5348	7083	88795	1097.42	0.0071	0.0	9.8e-06
la01_7_r_h	dual =	191800.0	opt =	193000.0	gap	= 0.0063	
State-Based	12765	18697	25842	3600.05	0.02143	0.0002176	0.006995
Break based	45455	12556	2724	3600.22	0.006484	0.0002176	0.006112
SCIP+	21347	891	1638	1100.25	0.005402	0.0	0.0
SCIP+: col. gen.	13933	964	6936	2963.05	0.0003365	0.0	0.0
Presolved break based	26529	12597	63330	931.83	0.0066	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la01_7_r_l	dual =	235000.0	opt =	236400.0	gap	= 0.0058	
State-Based	10328	18161	29265	3603.74	0.01278	6.346e-05	0.005675
Break based	26460	10207	13184	3600.53	0.005792	6.346e-05	0.00389
SCIP+	14702	777	430	232.95	0.004998	0.0	0.0
SCIP+: col. gen.	9004	898	768	212.5	0.001155	0.0	0.0
Presolved break based	17038	10258	93874	1281.44	0.0056	0.0	0.0
la01_7_r_m	dual =	160600.0	opt =	161600.0	gap	= 0.0064	
State-Based	9112	11836	23850	3600.05	0.01068	0.001411	0.005167
Break based	22670	9116	10422	3600.24	0.006123	0.001751	0.005558
SCIP+	7094	728	1362	496.27	0.005659	0.0	0.0
SCIP+: col. gen.	7740	767	8645	2078.76	0.0001086	0.0	0.0
Presolved break based	8126	9165	96543	1247.37	0.0061	0.0	4.9e-05
la01_7_r_s	dual =	257900.0	opt =	258800.0	gap	= 0.0034	
State-Based	6640	11716	31925	1243.67	0.006909	0.0	6.956e-05
Break based	10486	6756	22938	1003.72	0.003418	0.0	0.0
SCIP+	4203	606	1772	272.06	0.003344	0.0	0.0
SCIP+: col. gen.	4560	709	1756	236.82	0.0005788	0.0	0.0
Presolved break based	5338	6788	17151	163.13	0.0033	0.0	0.0
la01_7_s_h	dual =	83190.0	opt =	83970.0	gap	= 0.0093	
State-Based	13559	14552	6289	3600.16	0.02371	0.005657	0.02045
Break based	52818	13409	366	3600.71	0.009295	0.07136	0.08092
SCIP+	40208	925	476	3600.22	0.007798	0.006347	0.012
SCIP+: col. gen.	18367	930	3467	3600.02	0.0002269	0.0002501	0.004986
Presolved break based	46858	13456	56507	3600.39	0.0092	0.0	0.005
la01_7_s_l	dual =	87490.0	opt =	88130.0	gap	= 0.0072	
State-Based	11579	12482	3303	3600.01	0.02132	0.003529	0.02272
Break based	39357	11519	205	3600.21	0.007248	0.006071	0.01311
SCIP+	28564	835	743	3600.13	0.005061	0.001294	0.003238
SCIP+: col. gen.	14510	840	5113	3600.02	0.0009828	0.002281	0.004711
Presolved break based	34069	11565	28720	3600.04	0.007	0.0034	0.0076
la01_7_s_m	dual =	88530.0	opt =	88910.0	gap	= 0.0043	
State-Based	9599	10412	25279	3600.03	0.008958	0.002306	0.005506
Break based	25484	9648	365	3600.24	0.004338	0.002519	0.006584
SCIP+	18402	745	4154	1899.35	0.004447	0.0	0.0
SCIP+: col. gen.	10631	750	5688	3600.02	0.0001301	0.0	0.0007887
Presolved break based	21121	9694	22181	1000.8	0.0047	0.0	0.0
la01_7_s_s	dual =	88510.0	opt =	88910.0	gap	= 0.0045	
State-Based	7619	8342	29245	3600.05	0.008707	0.0	0.001149
Break based	15692	7767	6821	3600.15	0.004581	0.001451	0.004864
SCIP+	10053	655	2839	830.04	0.004359	0.0	0.0
SCIP+: col. gen.	6707	660	2068	798.67	0.000126	0.0	0.0
Presolved break based	12128	7817	25997	408.85	0.0045	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la01_8_l_h	dual =	627200.0	opt =	631400.0	gap	= 0.0066	
State-Based	12216	22358	19486	3600.05	0.03222	0.001403	0.02245
Break based	38238	12042	3285	2258.34	0.006628	0.0	0.0
SCIP+	8107	860	422	168.46	0.003011	0.0	0.0
SCIP+: col. gen.	7732	995	323	96.63	0.0002757	0.0	0.0
Presolved break based	10944	12085	6184	80.3	0.0068	0.0	0.0
la01_8_l_l	dual =	627600.0	opt =	631400.0	gap	= 0.006	
State-Based	10236	19118	28980	3600.24	0.03267	0.001403	0.02224
Break based	25019	10164	2900	1756.72	0.005936	0.0	0.0
SCIP+	6482	770	259	107.68	0.003155	0.0	0.0
SCIP+: col. gen.	6558	905	359	121.92	0.0002763	0.0	0.0
Presolved break based	6926	10211	467	75.04	0.0041	0.0	0.0
la01_8_l_m	dual =	629900.0	opt =	631400.0	gap	= 0.0023	
State-Based	8256	15878	26342	1784.46	0.02749	0.0	0.0
Break based	15825	8292	107	805.39	0.002233	0.0	6.336e-05
SCIP+	5009	680	97	31.2	0.00283	0.0	0.0
SCIP+: col. gen.	5446	815	370	82.21	0.000639	0.0	0.0
Presolved break based	5348	8342	176	46.34	0.0022	0.0	0.0
la01_8_l_s	dual =	634300.0	opt =	637700.0	gap	= 0.0053	
State-Based	6275	12637	48907	2684.9	0.02264	0.0	0.0
Break based	8630	6425	2430	661.81	0.005366	0.0	0.0
SCIP+	3755	590	1465	235.64	0.006243	0.0	0.0
SCIP+: col. gen.	4227	725	3606	597.9	0.0009119	0.0	0.0
Presolved break based	3958	6452	911	37.79	0.0054	0.0	0.0
la01_8_m_h	dual =	368700.0	opt =	378900.0	gap	= 0.027	
State-Based	12816	18968	14464	3600.04	0.0483	0.0	0.02922
Break based	44578	12673	361	3600.74	0.02681	0.01608	0.04252
SCIP+	10548	890	8053	2952.67	0.0204	0.0	0.0
SCIP+: col. gen.	9629	965	7498	2832.8	0.000678	0.0	0.0
Presolved break based	13957	12722	30366	517.39	0.026	0.0	0.0
la01_8_m_l	dual =	370200.0	opt =	378900.0	gap	= 0.023	
State-Based	10836	16268	18865	3600.1	0.04899	0.003555	0.0371
Break based	30087	10788	2829	3601.09	0.02297	0.0	0.02075
SCIP+	8172	800	5232	1840.39	0.01964	0.0	0.0
SCIP+: col. gen.	8162	875	7614	2493.54	0.0006794	0.0	0.0
Presolved break based	8944	10836	24090	673.26	0.023	0.0	0.0
la01_8_m_m	dual =	370200.0	opt =	378900.0	gap	= 0.023	
State-Based	8856	13568	29611	3600.27	0.04647	0.0	0.01076
Break based	19782	8922	7125	3600.41	0.02277	0.0	0.01235
SCIP+	6136	710	4483	1575.38	0.01987	0.0	0.0
SCIP+: col. gen.	6847	785	8019	2274.95	0.000941	0.0	0.0
Presolved break based	6708	8973	27130	447.45	0.022	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la01_8_m_s	dual =	371200.0	opt =	378900.0	gap	= 0.02	
State-Based	6876	10868	66159	3450.84	0.03631	0.0	0.0
Break based	11327	7042	39830	3600.7	0.02015	0.0	0.004348
SCIP+	4328	620	3058	880.08	0.01953	0.0	0.0
SCIP+: col. gen.	4785	695	6061	1369.92	0.001026	0.0	0.0
Presolved break based	4832	7083	26339	336.05	0.019	0.0	0.0
la01_8_r_h	dual =	339400.0	opt =	347500.0	gap	= 0.023	
State-Based	12795	18778	24868	3600.05	0.04344	0.0007913	0.01845
Break based	45092	12644	6017	3601.19	0.02327	0.0	0.01026
SCIP+	9927	891	8856	2457.45	0.01651	0.0	0.0
SCIP+: col. gen.	9938	964	2611	1084.87	0.001053	0.0	0.0
Presolved break based	13241	12687	16560	285.27	0.022	0.0	0.0
la01_8_r_l	dual =	300400.0	opt =	316900.0	gap	= 0.052	
State-Based	10560	16432	29221	3600.49	0.06556	0.001272	0.01548
Break based	29661	10447	605	3600.44	0.0519	0.0055	0.05297
SCIP+	7819	795	6910	3600.11	0.03301	0.0	0.005105
SCIP+: col. gen.	8229	880	5402	2082.06	0.0008821	0.0	0.0
Presolved break based	8150	10491	49570	1090.65	0.042	0.0	0.0
la01_8_r_m	dual =	342900.0	opt =	348400.0	gap	= 0.016	
State-Based	8542	13653	29371	3600.17	0.04947	0.003155	0.0145
Break based	19263	8537	6874	2429.85	0.01587	0.0	0.0
SCIP+	5970	705	4242	1377.43	0.01439	0.0	0.0
SCIP+: col. gen.	6539	790	1888	605.31	0.000804	0.0	0.0
Presolved break based	6582	8577	9949	200.65	0.015	0.0	0.0
la01_8_r_s	dual =	293400.0	opt =	294900.0	gap	= 0.0051	
State-Based	6772	10715	19881	1659.3	0.03648	0.0	7.118e-05
Break based	11312	6915	537	577.11	0.004999	0.0	0.0
SCIP+	4232	620	2004	341.52	0.004708	0.0	0.0
SCIP+: col. gen.	4614	695	2381	346.56	0.001104	0.0	0.0
Presolved break based	4596	6948	651	48.63	0.0049	0.0	0.0
la01_8_s_h	dual =	148200.0	opt =	151700.0	gap	= 0.023	
State-Based	13516	14553	29541	3600.09	0.03697	0.002155	0.01203
Break based	52630	13409	365	3601.07	0.02347	0.006438	0.02835
SCIP+	16049	925	2885	1811.83	0.018	0.0	0.0
SCIP+: col. gen.	12628	930	7397	3475.6	0.0007165	0.0	0.0
Presolved break based	18211	13459	130339	3313.48	0.023	0.0	0.0
la01_8_s_l	dual =	148200.0	opt =	151700.0	gap	= 0.023	
State-Based	11536	12483	30140	3600.28	0.0366	0.001021	0.01401
Break based	39090	11519	2921	3600.98	0.02324	0.01669	0.03862
SCIP+	12395	835	2557	1416.49	0.01769	0.0	0.0
SCIP+: col. gen.	10779	840	7990	3600.02	0.0007666	0.0	0.002843
Presolved break based	14504	11565	159938	2404.92	0.024	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la01_8_s_m	dual =	148700.0	opt =	151700.0	gap	= 0.02	
State-Based	9556	10413	34503	1960.21	0.03608	0.0	0.0
Break based	24863	9648	726	3600.5	0.01989	0.0	0.0187
SCIP+	8800	745	1790	1049.19	0.0179	0.0	0.0
SCIP+: col. gen.	8262	750	6102	2351.77	0.0007303	0.0	0.0
Presolved break based	9287	9701	95901	2041.23	0.02	0.0	0.0
la01_8_s_s	dual =	149200.0	opt =	151700.0	gap	= 0.017	
State-Based	7576	8343	10964	661.32	0.02983	0.0	0.0
Break based	14792	7767	20878	3600.24	0.01704	0.0	0.00564
SCIP+	5945	655	349	162.77	0.01543	0.0	0.0
SCIP+: col. gen.	5593	660	921	333.32	0.001404	0.0	0.0
Presolved break based	6351	7812	17357	289.33	0.017	0.0	0.0
la02_0_l_h	dual =	244400.0	opt =	249100.0	gap	= 0.019	
State-Based	10878	20027	9365	3600.04	0.04719	0.001016	0.02767
Break based	28331	10798	91	3600.39	0.01877	0.01312	0.0296
SCIP+	6282	790	3082	3600.12	0.01365	0.00939	0.01493
SCIP+: col. gen.	10070	925	4643	3162.07	0.0007846	0.0	0.0
Presolved break based	6723	10840	51992	1270.38	0.018	0.0	0.0
la02_0_l_l	dual =	271200.0	opt =	275100.0	gap	= 0.014	
State-Based	9115	17110	11505	3600.05	0.05798	0.004187	0.03643
Break based	19934	9145	344	3600.48	0.01417	0.01668	0.02976
SCIP+	5312	710	7921	3600.08	0.01319	0.0002799	0.0078
SCIP+: col. gen.	7458	845	7543	3433.07	0.001067	0.0	0.0
Presolved break based	5323	9190	123897	1692.42	0.014	0.0	0.0
la02_0_l_m	dual =	286300.0	opt =	287800.0	gap	= 0.0054	
State-Based	7244	14032	27266	3600.04	0.04652	0.0003057	0.02162
Break based	12386	7360	59069	2395.81	0.005196	0.0	0.0
SCIP+	4275	625	3611	477.63	0.004738	0.0	0.0
SCIP+: col. gen.	4963	760	1305	194.53	0.0007647	0.0	0.0
Presolved break based	4295	7417	29394	296.57	0.0051	0.0	0.0
la02_0_l_s	dual =	298200.0	opt =	299800.0	gap	= 0.0055	
State-Based	5534	11253	46724	3600.09	0.02227	0.0	0.00658
Break based	6712	5689	31724	1656.81	0.005484	0.0	0.0
SCIP+	3279	543	517	104.98	0.005466	0.0	0.0
SCIP+: col. gen.	3673	680	1067	213.83	0.0006291	0.0	0.0
Presolved break based	3318	5725	13375	115.57	0.0054	0.0	0.0
la02_0_m_h	dual =	145400.0	opt =	148600.0	gap	= 0.022	
State-Based	11473	17000	13775	3600.04	0.03614	0.002254	0.02267
Break based	36213	11425	343	3600.13	0.02174	0.02469	0.04639
SCIP+	8751	820	3615	3600.13	0.01757	0.004731	0.008908
SCIP+: col. gen.	12254	895	4201	3600.02	0.001014	0.0	0.005862
Presolved break based	12480	11472	73894	1125.4	0.021	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la02_0_m_l	dual =	162800.0	opt =	165500.0	gap	= 0.017	
State-Based	9716	14636	15468	3600.04	0.04807	0.00775	0.03154
Break based	24507	9766	549	3600.18	0.01672	0.003866	0.01982
SCIP+	6621	740	6140	3600.08	0.01632	0.0004893	0.003002
SCIP+: col. gen.	9792	815	5618	3600.02	0.0007362	0.0004893	0.006427
Presolved break based	7442	9816	164322	2719.88	0.017	0.0	3.6e-05
la02_0_m_m	dual =	175400.0	opt =	177000.0	gap	= 0.009	
State-Based	7842	12049	20618	3600.05	0.04783	0.0002543	0.02027
Break based	15948	7990	16277	3600.16	0.009234	0.001141	0.005933
SCIP+	4646	655	19238	3600.05	0.008333	0.0	0.0004153
SCIP+: col. gen.	5935	730	4622	1496.24	0.0006291	0.0	0.0
Presolved break based	5257	8042	37737	406.1	0.0092	0.0	4.5e-05
la02_0_m_s	dual =	182800.0	opt =	184300.0	gap	= 0.0081	
State-Based	6089	9688	44473	3600.06	0.02836	0.001688	0.006099
Break based	9327	6317	50281	3600.8	0.008083	2.713e-05	0.00366
SCIP+	3676	575	7091	1434.58	0.007551	0.0	0.0
SCIP+: col. gen.	4197	650	8964	2032.56	0.0001958	0.0	0.0
Presolved break based	3693	6364	83452	512.7	0.0083	0.0	0.0
la02_0_r_h	dual =	181300.0	opt =	185000.0	gap	= 0.02	
State-Based	11232	16151	30350	3600.15	0.03724	0.001178	0.01039
Break based	34192	11119	2117	3600.17	0.02017	0.006826	0.02525
SCIP+	15369	824	3811	3600.14	0.02023	0.005891	0.0199
SCIP+: col. gen.	14424	891	8887	3600.02	0.001239	0.001113	0.01463
Presolved break based	17562	11169	32985	1200.7	0.02	0.0	0.0
la02_0_r_l	dual =	159500.0	opt =	161200.0	gap	= 0.01	
State-Based	9793	14575	19016	3600.02	0.0364	0.005038	0.01794
Break based	25219	9846	114	3600.65	0.01031	0.01642	0.02532
SCIP+	6650	740	4414	3600.1	0.007397	0.0	0.002376
SCIP+: col. gen.	9277	815	5943	2886.8	0.0003592	0.0	0.0
Presolved break based	7706	9895	127467	2828.8	0.01	0.0	0.0
la02_0_r_m	dual =	141700.0	opt =	144000.0	gap	= 0.016	
State-Based	7563	11913	26913	3600.03	0.0567	0.003446	0.02187
Break based	15571	7653	21721	3600.17	0.01554	0.001389	0.007464
SCIP+	4942	653	877	196.6	0.01333	0.0	0.0
SCIP+: col. gen.	7469	732	6399	2141.25	0.0005101	0.0	0.0
Presolved break based	6369	7706	33281	675.31	0.011	0.0	0.0
la02_0_r_s	dual =	149300.0	opt =	150300.0	gap	= 0.0064	
State-Based	6014	8934	72472	2576.27	0.02114	0.0	0.0
Break based	10010	6182	17692	1649.32	0.006464	0.0	0.0
SCIP+	4735	581	3975	1037.98	0.005046	0.0	0.0
SCIP+: col. gen.	4840	644	2102	531.43	0.000411	0.0	0.0
Presolved break based	5102	6222	17787	190.91	0.0063	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la02_0_s_h	dual =	58640.0	opt =	59930.0	gap	= 0.022	
State-Based	12173	13077	20593	3600.06	0.03661	0.0	0.01977
Break based	43464	12160	3281	3600.17	0.02161	0.0	0.01818
SCIP+	15933	855	3516	3600.17	0.01801	0.004455	0.02229
SCIP+: col. gen.	14964	860	4755	3600.02	0.0009692	0.009444	0.02382
Presolved break based	20094	12208	114292	3600.27	0.021	0.0	0.0059
la02_0_s_l	dual =	66370.0	opt =	67600.0	gap	= 0.018	
State-Based	10416	11252	25884	3600.04	0.03931	0.003003	0.02567
Break based	30487	10489	1794	3600.56	0.01823	0.003373	0.01959
SCIP+	11895	775	3642	3600.12	0.01597	0.006597	0.02218
SCIP+: col. gen.	13453	780	4680	3600.02	0.0005613	0.006642	0.01947
Presolved break based	13139	10535	136187	3487.34	0.018	0.0	0.0
la02_0_s_m	dual =	72300.0	opt =	73280.0	gap	= 0.013	
State-Based	8542	9281	29279	3600.03	0.04309	0.001351	0.01846
Break based	20750	8725	2908	3600.17	0.01346	0.01661	0.02893
SCIP+	7559	690	3134	1442.46	0.01092	0.0	0.0
SCIP+: col. gen.	9728	695	5216	3162.76	0.0004464	0.0	0.0
Presolved break based	8628	8774	85479	1262.23	0.013	0.0	5.5e-05
la02_0_s_s	dual =	76710.0	opt =	77210.0	gap	= 0.0064	
State-Based	6785	7456	45839	3600.1	0.02858	0.001023	0.006014
Break based	12941	7048	55637	3612.36	0.006376	0.0	0.001531
SCIP+	4697	610	2239	458.88	0.005482	0.0	0.0
SCIP+: col. gen.	5183	615	5103	932.09	0.0001128	0.0	0.0
Presolved break based	5151	7099	66913	594.57	0.0061	0.0	0.0
la02_1_l_h	dual =	6006.0	opt =	6006.0	gap	= -0.0	
State-Based	10875	19973	1380	1414.79	0.0	0.0	0.0
Break based	28087	10798	1	3172.15	0.0	0.0	0.0
SCIP+	9085	790	184	701.76	0.0	0.0	0.0
SCIP+: col. gen.	8021	925	351	595.49	0.0	0.0	0.0
Presolved break based	10366	10845	1	345.21	0.0	0.0	0.0
la02_1_l_l	dual =	6006.0	opt =	6006.0	gap	= 0.0	
State-Based	9114	17092	1	153.16	0.0	0.0	0.0
Break based	19690	9145	1	2016.37	0.0	0.0	0.0
SCIP+	6789	710	167	437.65	0.0	0.0	0.0
SCIP+: col. gen.	6561	845	380	436.31	0.0	0.0	0.0
Presolved break based	7849	9186	1	129.89	0.0	0.0	0.0
la02_1_l_m	dual =	6006.0	opt =	6006.0	gap	= 0.0	
State-Based	7244	14032	757	486.93	0.0	0.0	0.0
Break based	12148	7360	1	953.7	0.0	0.0	0.0
SCIP+	4630	625	196	360.0	0.0	0.0	0.0
SCIP+: col. gen.	4977	760	530	537.81	0.0	0.0	0.0
Presolved break based	5554	7401	1	36.02	0.0	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la02_1_l_s	dual =	6006.0	opt =	6006.0	gap	= -0.0	
State-Based	5531	11199	5977	1135.03	0.0	0.0	0.0
Break based	6402	5671	428	3600.32	0.0	0.0	0.0
SCIP+	3350	543	3884	3369.99	0.0	0.0	0.0
SCIP+: col. gen.	4223	680	1893	1909.92	0.0	0.0	0.0
Presolved break based	3638	5721	331	136.82	0.0	0.0	0.0
la02_1_m_h	dual =	3759.0	opt =	3759.0	gap	= -0.0	
State-Based	11473	17000	2288	1939.99	0.0	0.0	0.0
Break based	36037	11425	1	3600.35	0.0	0.002394	0.002394
SCIP+	14877	820	1242	1554.5	0.0	0.0	0.0
SCIP+: col. gen.	8431	895	316	285.37	0.0	0.0	0.0
Presolved break based	21197	11478	115	1045.6	0.0	0.0	0.0
la02_1_m_l	dual =	3759.0	opt =	3759.0	gap	= -0.0	
State-Based	9713	14600	1	162.16	0.0	0.0	0.0
Break based	24321	9766	1	2089.33	0.0	0.0	0.0
SCIP+	10179	740	187	433.57	0.0	0.0	0.0
SCIP+: col. gen.	6828	815	530	481.22	0.0	0.0	0.0
Presolved break based	12561	9815	1	591.23	0.0	0.0	0.0
la02_1_m_m	dual =	3759.0	opt =	3759.0	gap	= 0.0	
State-Based	7842	12049	2336	449.66	0.0	0.0	0.0
Break based	15762	7990	1	1200.14	0.0	0.0	0.0
SCIP+	6592	655	76	176.13	0.0	0.0	0.0
SCIP+: col. gen.	5502	730	303	350.28	0.0	0.0	0.0
Presolved break based	8100	8034	1	50.89	0.0	0.0	0.0
la02_1_m_s	dual =	3759.0	opt =	3759.0	gap	= 0.0	
State-Based	6086	9652	848	239.15	0.0	0.0	0.0
Break based	9035	6307	3774	1596.45	0.0	0.0	0.0
SCIP+	4440	575	46	66.01	0.0	0.0	0.0
SCIP+: col. gen.	4584	650	434	602.47	0.0	0.0	0.0
Presolved break based	5177	6354	256	454.28	0.0	0.0	0.0
la02_1_r_h	dual =	3362.0	opt =	3362.0	gap	= 0.0	
State-Based	11503	16399	1	195.76	0.0	0.0	0.0
Break based	36502	11436	1	3600.2	0.0	0.02499	0.02499
SCIP+	13818	824	1567	1913.23	0.0	0.0	0.0
SCIP+: col. gen.	7899	891	175	226.17	0.0	0.0	0.0
Presolved break based	19175	11489	1	257.04	0.0	0.0	0.0
la02_1_r_l	dual =	3581.0	opt =	3581.0	gap	= -0.0	
State-Based	10048	12740	1526	759.41	0.0	0.0	0.0
Break based	27537	10107	1	2287.72	0.0	0.0	0.0
SCIP+	16325	759	163	902.99	0.0	0.0	0.0
SCIP+: col. gen.	9611	796	595	1175.23	0.0	0.0	0.0
Presolved break based	18809	10158	1	224.62	0.0	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la02_1_r_m	dual =	3788.0	opt =	3788.0	gap	= 0.0	
State-Based	7326	12628	1301	308.68	0.0	0.0	0.0
Break based	14090	7370	1	788.51	0.0	0.0	0.0
SCIP+	6938	641	93	278.73	0.0	0.0	0.0
SCIP+: col. gen.	5575	744	762	769.84	0.0	0.0	0.0
Presolved break based	8856	7409	1	31.02	0.0	0.0	0.0
la02_1_r_s	dual =	4229.0	opt =	4229.0	gap	= -0.0	
State-Based	6202	8482	15101	3196.24	0.0	0.0	0.0
Break based	10427	6344	115	3600.16	0.0	0.0004729	0.0004729
SCIP+	5812	591	404	604.55	0.0	0.0	0.0
SCIP+: col. gen.	4749	634	380	384.31	0.0	0.0	0.0
Presolved break based	7676	6391	439	1103.6	0.0	0.0	0.0
la02_1_s_h	dual =	1662.0	opt =	1662.0	gap	= 0.0	
State-Based	12173	13077	119	463.12	0.0	0.0	0.0
Break based	43358	12160	115	3600.62	0.0	0.0006017	0.0006017
SCIP+	29230	855	402	2656.05	0.0	0.0	0.0
SCIP+: col. gen.	9794	860	205	489.08	0.0	0.0	0.0
Presolved break based	35736	12213	1	542.84	0.0	0.0	0.0
la02_1_s_l	dual =	1662.0	opt =	1662.0	gap	= 0.0	
State-Based	10413	11237	4060	2128.61	0.0	0.0	0.0
Break based	30371	10489	1	2858.79	0.0	0.0	0.0
SCIP+	20150	775	535	1899.06	0.0	0.0	0.0
SCIP+: col. gen.	8773	780	231	680.0	0.0	0.0	0.0
Presolved break based	23511	10542	1	505.87	0.0	0.0	0.0
la02_1_s_m	dual =	1662.0	opt =	1662.0	gap	= 0.0	
State-Based	8542	9281	2804	254.48	0.0	0.0	0.0
Break based	20634	8725	1	3293.32	0.0	0.0	0.0
SCIP+	12445	690	67	193.79	0.0	0.0	0.0
SCIP+: col. gen.	6779	695	364	488.95	0.0	0.0	0.0
Presolved break based	14756	8776	1	101.96	0.0	0.0	0.0
la02_1_s_s	dual =	1662.0	opt =	1662.0	gap	= 0.0	
State-Based	6782	7441	1	106.45	0.0	0.0	0.0
Break based	12770	7045	1	934.83	0.0	0.0	0.0
SCIP+	6765	610	51	113.3	0.0	0.0	0.0
SCIP+: col. gen.	4881	615	179	171.66	0.0	0.0	0.0
Presolved break based	8551	7098	1	117.55	0.0	0.0	0.0
la02_7_l_h	dual =	334100.0	opt =	335800.0	gap	= 0.005	
State-Based	10877	20009	10739	3600.06	0.02387	0.005066	0.02131
Break based	28331	10798	987	3600.17	0.004947	0.006117	0.01044
SCIP+	7161	790	7301	3600.09	0.00426	0.0	0.0001658
SCIP+: col. gen.	8478	925	9254	3600.02	0.0001024	0.0	0.0008736
Presolved break based	8007	10841	79712	1210.9	0.0048	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la02_7_l_l	dual =	339500.0	opt =	342500.0	gap	= 0.0088	
State-Based	9114	17092	27036	3600.04	0.02271	0.003924	0.0154
Break based	19931	9145	18819	3600.7	0.008693	0.001524	0.004763
SCIP+	5434	710	538	246.55	0.006526	0.0	0.0
SCIP+: col. gen.	6109	845	1606	355.2	0.0	0.0	0.0
Presolved break based	6150	9189	78663	954.73	0.0086	0.0	0.0
la02_7_l_m	dual =	341900.0	opt =	344000.0	gap	= 0.006	
State-Based	7244	14032	29791	3600.02	0.01203	4.361e-05	0.007786
Break based	12386	7360	54977	3604.63	0.006029	0.0	0.001255
SCIP+	4322	625	3521	803.02	0.004848	0.0	0.0
SCIP+: col. gen.	4762	760	11438	2061.69	0.0003743	0.0	0.0
Presolved break based	4400	7408	94875	932.83	0.0059	0.0	8.7e-05
la02_7_l_s	dual =	343800.0	opt =	346200.0	gap	= 0.0069	
State-Based	5536	11289	68825	3600.04	0.01156	0.0	0.00178
Break based	6678	5685	49505	2303.46	0.006998	0.0	8.947e-05
SCIP+	3302	543	6944	1583.23	0.005902	0.0	0.0
SCIP+: col. gen.	3812	680	4403	1020.84	0.001167	0.0	0.0
Presolved break based	3340	5722	25816	274.65	0.007	0.0	0.0
la02_7_m_h	dual =	209900.0	opt =	211600.0	gap	= 0.0082	
State-Based	11475	17003	10863	3600.07	0.02593	0.003657	0.02486
Break based	36213	11425	1077	3600.6	0.00806	0.004021	0.01123
SCIP+	12060	820	2010	2409.88	0.0057	0.0	0.0
SCIP+: col. gen.	11332	895	4726	2999.19	0.0001585	0.0	0.0
Presolved break based	16952	11472	28189	749.53	0.008	0.0	0.0
la02_7_m_l	dual =	212600.0	opt =	213900.0	gap	= 0.006	
State-Based	9718	14639	25319	3600.03	0.01533	0.00397	0.01137
Break based	24507	9766	8380	3600.56	0.006194	0.0002244	0.005466
SCIP+	7995	740	5809	1977.05	0.004901	0.0	0.0
SCIP+: col. gen.	8085	815	2166	1163.02	0.0	0.0	0.0
Presolved break based	9864	9818	12170	423.4	0.0056	0.0	0.0
la02_7_m_m	dual =	212800.0	opt =	213900.0	gap	= 0.005	
State-Based	7844	12052	28981	3600.17	0.009617	0.0	0.002099
Break based	15948	7990	19804	3600.9	0.004918	0.0	0.000435
SCIP+	5363	655	866	266.54	0.003856	0.0	0.0
SCIP+: col. gen.	5519	730	1292	544.95	0.0005266	0.0	0.0
Presolved break based	6263	8033	11202	295.43	0.0049	0.0	0.0
la02_7_m_s	dual =	213400.0	opt =	214800.0	gap	= 0.0063	
State-Based	6090	9680	63052	3600.08	0.009022	0.0004703	0.00368
Break based	9327	6317	26990	2091.12	0.006381	0.0	0.0
SCIP+	3712	575	4900	1147.07	0.006321	0.0	0.0
SCIP+: col. gen.	4214	650	7075	1950.02	0.0008492	0.0	0.0
Presolved break based	4101	6363	16540	248.3	0.0064	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la02_7_r_h	dual =	214900.0	opt =	216900.0	gap	= 0.0092	
State-Based	11462	15111	3668	3600.09	0.02604	0.0113	0.03357
Break based	38156	11366	276	3600.67	0.009304	0.01209	0.02061
SCIP+	15718	834	560	776.22	0.005991	0.0	0.0
SCIP+: col. gen.	11172	881	2174	1506.9	0.000254	0.0	0.0
Presolved break based	20417	11422	44677	2472.17	0.0086	0.0	0.0
la02_7_r_l	dual =	255800.0	opt =	257900.0	gap	= 0.0082	
State-Based	9100	15904	27010	3600.07	0.02021	0.002621	0.01335
Break based	21596	9113	2859	3600.27	0.008166	0.001353	0.008863
SCIP+	9686	722	7878	2194.97	0.007731	0.0	0.0
SCIP+: col. gen.	8279	834	11814	2967.0	0.0	0.0	0.0
Presolved break based	11552	9160	160563	1487.05	0.0081	0.0	8.1e-05
la02_7_r_m	dual =	235100.0	opt =	236500.0	gap	= 0.0058	
State-Based	7591	12283	28719	1876.98	0.01226	0.0	0.0
Break based	15574	7674	15001	3600.33	0.005774	0.0	0.001911
SCIP+	4534	649	3405	1072.3	0.004568	0.0	0.0
SCIP+: col. gen.	5243	738	16501	3600.01	0.0002355	0.0	0.001445
Presolved break based	6330	7714	27231	324.37	0.0058	0.0	0.0
la02_7_r_s	dual =	200600.0	opt =	202200.0	gap	= 0.0078	
State-Based	5578	9958	29872	3600.11	0.01111	0.001523	0.007368
Break based	8206	5747	33556	3600.45	0.007666	0.0008458	0.00384
SCIP+	3449	563	8057	2573.8	0.006953	0.0	0.0
SCIP+: col. gen.	4803	662	10846	3600.01	0.0002521	0.001523	0.003799
Presolved break based	5321	5777	30261	446.26	0.0077	0.0	9.4e-05
la02_7_s_h	dual =	89910.0	opt =	91590.0	gap	= 0.018	
State-Based	12175	13078	12735	3600.1	0.03478	0.01991	0.04429
Break based	43464	12160	969	3600.81	0.01832	0.1196	0.1389
SCIP+	31279	855	80	3600.18	0.01301	0.01545	0.024
SCIP+: col. gen.	16476	860	3613	3600.02	0.0	0.008691	0.01811
Presolved break based	37119	12213	56268	3603.16	0.017	0.0027	0.013
la02_7_s_l	dual =	91580.0	opt =	92440.0	gap	= 0.0093	
State-Based	10415	11238	26852	3600.06	0.01513	0.001136	0.008533
Break based	30487	10489	89	3600.69	0.009288	0.001785	0.009957
SCIP+	22437	775	7296	3333.91	0.008302	0.0	0.0
SCIP+: col. gen.	14862	780	6763	3600.02	0.0001033	0.003862	0.006879
Presolved break based	25339	10543	23960	1781.96	0.0088	0.0	0.0
la02_7_s_m	dual =	91760.0	opt =	92540.0	gap	= 0.0085	
State-Based	8544	9282	29632	3600.12	0.0127	0.0008104	0.008131
Break based	20750	8725	498	3600.32	0.00843	0.007899	0.01474
SCIP+	14235	690	9374	3600.08	0.007277	0.0	0.003276
SCIP+: col. gen.	11089	695	7654	3600.01	0.0001697	0.0	0.001261
Presolved break based	16567	8778	14139	1033.49	0.008	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la02_7_s_s	dual =	91950.0	opt =	92850.0	gap	= 0.0097	
State-Based	6784	7442	52243	3616.54	0.01448	0.0005708	0.006118
Break based	12941	7048	6745	3600.51	0.00971	0.003457	0.01101
SCIP+	7787	610	11307	3600.04	0.009142	0.000797	0.004449
SCIP+: col. gen.	7272	615	10303	3600.02	0.0004436	0.00028	0.003661
Presolved break based	9608	7099	23942	664.23	0.0096	0.0	0.0
la02_8_l_h	dual =	573100.0	opt =	580800.0	gap	= 0.013	
State-Based	10832	19974	16708	3600.06	0.03875	0.006887	0.02868
Break based	28224	10798	3764	3600.7	0.01339	0.006599	0.01686
SCIP+	6777	790	6878	3600.1	0.01278	0.0008901	0.005317
SCIP+: col. gen.	7569	925	2958	1425.1	0.001698	0.0	0.0
Presolved break based	7180	10834	23058	650.6	0.013	0.0	0.0
la02_8_l_l	dual =	572800.0	opt =	580800.0	gap	= 0.014	
State-Based	9071	17093	25194	3600.06	0.03813	0.0	0.01286
Break based	19769	9145	12897	3600.64	0.01388	0.001672	0.006954
SCIP+	5392	710	7088	3600.06	0.01283	0.008212	0.01285
SCIP+: col. gen.	6172	845	2801	1181.59	0.001693	0.0	0.0
Presolved break based	5835	9190	39843	596.18	0.013	0.0	0.0
la02_8_l_m	dual =	577900.0	opt =	589500.0	gap	= 0.02	
State-Based	7201	14033	35739	3600.09	0.0301	0.0	0.0129
Break based	12096	7360	14812	3600.31	0.0197	0.001778	0.01024
SCIP+	4276	625	9369	3600.04	0.01921	0.0003037	0.003805
SCIP+: col. gen.	5315	760	8646	3600.02	0.00313	0.0003037	0.008585
Presolved break based	4409	7408	38720	549.01	0.02	0.0	0.0
la02_8_l_s	dual =	592900.0	opt =	605400.0	gap	= 0.021	
State-Based	5493	11290	34675	3600.06	0.03395	0.0	0.007839
Break based	6558	5689	21536	3600.2	0.02057	0.000114	0.001851
SCIP+	3268	541	4631	1701.37	0.02023	0.000114	0.0
SCIP+: col. gen.	4520	680	9287	3600.01	0.003909	0.000114	0.004885
Presolved break based	3321	5724	17836	335.99	0.021	0.00011	0.0
la02_8_m_h	dual =	336000.0	opt =	341200.0	gap	= 0.015	
State-Based	11432	17004	13495	3600.06	0.04553	0.0001935	0.03287
Break based	36090	11425	3362	3600.14	0.01511	0.0008617	0.01297
SCIP+	9834	820	4998	3600.13	0.01216	0.01059	0.0156
SCIP+: col. gen.	9311	895	5621	3188.56	0.0009104	0.0	0.0
Presolved break based	13278	11475	63165	836.15	0.014	0.0	0.0
la02_8_m_l	dual =	336400.0	opt =	341200.0	gap	= 0.014	
State-Based	9672	14604	25667	3600.03	0.04498	0.0	0.006001
Break based	24288	9766	757	3600.35	0.01406	0.00345	0.01511
SCIP+	7343	740	7938	3600.13	0.01173	0.0	0.002115
SCIP+: col. gen.	7358	815	5704	2675.48	0.0009521	0.0	0.0
Presolved break based	8532	9822	79154	1112.68	0.013	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la02_8_m_m	dual =	336900.0	opt =	341200.0	gap	= 0.013	
State-Based	7801	12053	53231	2474.24	0.03568	0.0	0.0
Break based	15538	7990	29462	3319.06	0.01246	0.0	0.0
SCIP+	5287	655	4406	1483.23	0.01192	0.0	0.0
SCIP+: col. gen.	5906	730	5137	1981.33	0.001095	0.0	0.0
Presolved break based	6022	8045	32025	363.75	0.012	0.0	0.0
la02_8_m_s	dual =	340000.0	opt =	344400.0	gap	= 0.013	
State-Based	6049	9705	68631	3600.07	0.03224	0.0	0.002664
Break based	9009	6317	13861	3600.31	0.01256	0.0	0.003646
SCIP+	3726	575	11910	3302.99	0.0151	0.0	0.0
SCIP+: col. gen.	4437	650	10322	3600.01	0.001556	0.0	0.004103
Presolved break based	4134	6360	16199	266.08	0.013	0.0	0.0
la02_8_r_h	dual =	288400.0	opt =	292700.0	gap	= 0.015	
State-Based	11336	16180	39463	3600.13	0.02942	0.0	0.004375
Break based	34650	11310	6875	3600.25	0.01447	0.0	0.003036
SCIP+	12422	824	12461	3600.14	0.0135	0.0	0.01245
SCIP+: col. gen.	10578	891	13687	3600.01	0.0007297	0.0	0.006177
Presolved break based	12970	11359	19136	283.64	0.013	0.0	0.0
la02_8_r_l	dual =	194700.0	opt =	195600.0	gap	= 0.0045	
State-Based	9483	14035	5750	761.04	0.02219	0.0	0.0
Break based	24213	9536	304	1574.3	0.004389	0.0	0.0
SCIP+	9660	742	91	58.33	0.004916	0.0	0.0
SCIP+: col. gen.	6566	813	137	65.49	0.0009036	0.0	0.0
Presolved break based	11343	9595	641	99.85	0.005	0.0	0.0
la02_8_r_m	dual =	305700.0	opt =	310000.0	gap	= 0.014	
State-Based	8030	11119	13346	639.41	0.04999	0.0	0.0
Break based	17408	8199	6455	2473.42	0.01375	0.0	0.0
SCIP+	6260	665	2368	768.52	0.01293	0.0	0.0
SCIP+: col. gen.	6642	720	11288	3600.01	0.00187	0.0	0.005126
Presolved break based	6380	8248	3171	115.79	0.011	0.0	0.0
la02_8_r_s	dual =	345200.0	opt =	351200.0	gap	= 0.017	
State-Based	5858	9555	80654	3600.13	0.0284	0.001751	0.00479
Break based	8636	6106	20915	2705.98	0.01697	0.0	0.0
SCIP+	4435	574	12413	3103.92	0.01577	0.0	0.0
SCIP+: col. gen.	4833	654	12944	3600.01	0.003704	0.001313	0.007817
Presolved break based	5355	6141	43536	525.29	0.017	0.0	0.0
la02_8_s_h	dual =	136900.0	opt =	140200.0	gap	= 0.024	
State-Based	12130	13078	27305	3600.04	0.03971	0.01714	0.04199
Break based	43200	12160	154	3600.25	0.02354	0.02241	0.04694
SCIP+	15507	855	4957	2803.46	0.01864	0.0	0.0
SCIP+: col. gen.	11541	860	8134	3600.02	0.0002435	0.004301	0.009823
Presolved break based	17620	12212	52059	1434.07	0.021	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la02_8_s_l	dual =	137200.0	opt =	140200.0	gap	= 0.021	
State-Based	10370	11238	30173	3600.09	0.03931	0.006056	0.02887
Break based	30085	10489	5331	3600.4	0.02174	0.02773	0.04832
SCIP+	12030	775	4027	2116.33	0.01788	0.0	0.0
SCIP+: col. gen.	9959	780	7540	3600.02	0.0002382	0.0002996	0.006113
Presolved break based	12053	10541	49419	1619.22	0.019	0.0	0.0
la02_8_s_m	dual =	137500.0	opt =	140900.0	gap	= 0.024	
State-Based	8499	9282	54859	3600.11	0.04043	0.0	0.003389
Break based	19908	8725	4474	3600.32	0.02449	0.0	0.01978
SCIP+	8199	690	3832	2224.08	0.02177	0.0	0.0
SCIP+: col. gen.	7411	695	6555	2986.37	0.0003454	0.0	0.0
Presolved break based	8641	8779	50471	845.27	0.022	0.0	0.0
la02_8_s_s	dual =	138300.0	opt =	140900.0	gap	= 0.019	
State-Based	6740	7447	11385	657.27	0.03787	0.0	0.0
Break based	12149	7048	21134	3159.27	0.01895	0.0	0.0
SCIP+	5366	610	1269	593.46	0.01802	0.0	0.0
SCIP+: col. gen.	5802	615	7022	3302.0	0.000442	0.0	0.0
Presolved break based	5907	7101	12675	308.55	0.018	0.0	0.0
la03_0_l_h	dual =	248800.0	opt =	252100.0	gap	= 0.013	
State-Based	9833	18185	16419	3600.04	0.04133	0.000718	0.01877
Break based	23936	9856	4844	3603.8	0.01314	0.00881	0.02131
SCIP+	6854	735	5864	3600.09	0.008594	0.0005276	0.002005
SCIP+: col. gen.	8411	870	5768	1718.0	0.0007523	0.0001428	0.0
Presolved break based	7857	9885	24900	639.58	0.013	0.0	0.0
la03_0_l_l	dual =	269500.0	opt =	272800.0	gap	= 0.012	
State-Based	8186	15539	25272	3600.04	0.04785	0.003431	0.03206
Break based	16727	8287	1093	3600.38	0.01199	0.01259	0.02196
SCIP+	5064	660	4550	1020.75	0.007992	0.0	0.0
SCIP+: col. gen.	6376	795	8782	2218.99	0.0009134	0.0	0.0
Presolved break based	6032	8320	38216	642.83	0.011	0.0	0.0
la03_0_l_m	dual =	283600.0	opt =	284600.0	gap	= 0.0034	
State-Based	6421	12569	30087	3600.07	0.03205	0.001529	0.007078
Break based	10329	6611	20658	1474.7	0.003272	0.0	0.0
SCIP+	3873	580	1337	198.83	0.002533	0.0	0.0
SCIP+: col. gen.	4776	715	11237	1093.25	0.0003556	0.0	0.0
Presolved break based	4243	6643	20565	193.95	0.0034	0.0	6.6e-05
la03_0_l_s	dual =	∞	opt =	-∞	gap	= 0.0	
State-Based	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SCIP+	3159	685	0.0	0.03	0.0	0.0	0.0
SCIP+: col. gen.	3296	640	1	0.23	0.0	0.0	0.0
Presolved break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la03_0_m_h	dual =	151700.0	opt =	154500.0	gap	= 0.018	
State-Based	10425	15501	21900	3600.06	0.0337	0.002537	0.01904
Break based	28970	10465	236	3600.78	0.01831	0.02066	0.03777
SCIP+	9827	765	2880	3600.12	0.01473	0.001527	0.008498
SCIP+: col. gen.	11362	840	10560	3600.02	0.001192	0.001877	0.007784
Presolved break based	11379	10499	201872	3600.18	0.019	0.0	0.0015
la03_0_m_l	dual =	166100.0	opt =	168400.0	gap	= 0.014	
State-Based	8777	13275	17364	3600.05	0.03999	0.001604	0.01535
Break based	20882	8917	14797	3600.33	0.01333	0.00427	0.01409
SCIP+	7272	690	4702	3600.07	0.01081	0.003789	0.009277
SCIP+: col. gen.	8594	765	9630	3600.02	0.0009805	0.0	0.002151
Presolved break based	8331	8949	201871	3600.09	0.013	0.0	0.0021
la03_0_m_m	dual =	176500.0	opt =	177600.0	gap	= 0.0063	
State-Based	7015	10851	30100	3600.15	0.03071	0.0	0.005115
Break based	13466	7237	30673	3128.57	0.006361	0.0	0.0
SCIP+	5096	610	9763	3600.04	0.003963	0.004808	0.005639
SCIP+: col. gen.	5375	685	3236	520.26	0.000895	0.0	0.0
Presolved break based	5933	7274	75262	598.59	0.006	0.0	0.0
la03_0_m_s	dual =	181900.0	opt =	183300.0	gap	= 0.0078	
State-Based	5380	8616	78452	3600.1	0.01813	0.0009491	0.003884
Break based	7820	5668	22406	1806.53	0.007631	0.0	0.0
SCIP+	3358	535	2990	661.18	0.005565	0.0	0.0
SCIP+: col. gen.	3899	610	3274	575.12	0.0	0.0	0.0
Presolved break based	4114	5691	14309	189.16	0.0076	0.0	0.0
la03_0_r_h	dual =	90670.0	opt =	91530.0	gap	= 0.0094	
State-Based	10536	15083	14745	3600.06	0.02604	0.0008849	0.007955
Break based	30174	10592	248	3600.62	0.009439	0.15	0.1607
SCIP+	11921	769	4035	3600.12	0.009363	0.0	0.004592
SCIP+: col. gen.	9278	836	1420	686.26	0.0003037	0.0	0.0
Presolved break based	13903	10626	19493	670.21	0.01	0.0	0.0
la03_0_r_l	dual =	139700.0	opt =	141500.0	gap	= 0.013	
State-Based	8677	11971	30028	3607.85	0.03314	0.001795	0.009053
Break based	22755	8753	593	3600.25	0.01321	0.01091	0.02186
SCIP+	10647	701	5042	3600.09	0.01137	0.002282	0.01042
SCIP+: col. gen.	10004	754	13646	3600.03	0.001391	0.00106	0.009213
Presolved break based	11985	8783	50142	1087.88	0.014	0.0	0.0
la03_0_r_m	dual =	162400.0	opt =	162700.0	gap	= 0.0018	
State-Based	6768	10860	10432	638.56	0.01486	0.0	5.532e-05
Break based	12772	6952	5357	782.74	0.002042	0.0	0.0
SCIP+	4011	604	681	101.72	0.001324	0.0	0.0
SCIP+: col. gen.	4459	691	1281	102.71	0.0006312	0.0001291	0.0
Presolved break based	4128	6991	6365	54.2	0.002	0.0	4.3e-05

instance	vars	cons	nodes	time	relative root	relative primal	gap
la03_0_r_s	dual =	170100.0	opt =	170800.0	gap	= 0.0041	
State-Based	5211	8528	71022	2806.16	0.01582	0.0	0.0
Break based	7706	5349	869	764.95	0.004229	0.0	0.0
SCIP+	3151	534	673	152.48	0.004661	0.0	0.0
SCIP+: col. gen.	3652	613	1605	326.8	0.0005803	0.0	0.0
Presolved break based	3499	5378	1783	54.13	0.0048	0.0	8.8e-05
la03_0_s_h	dual =	61350.0	opt =	62050.0	gap	= 0.011	
State-Based	11115	11944	27248	3600.03	0.01831	0.004062	0.01303
Break based	37920	11198	4776	3600.48	0.01119	0.00353	0.01272
SCIP+	15602	800	2556	3600.14	0.01135	0.0	0.006961
SCIP+: col. gen.	13160	805	7057	3600.02	0.0002667	0.0	0.006112
Presolved break based	19504	11230	38586	830.57	0.011	0.0	4.8e-05
la03_0_s_l	dual =	67670.0	opt =	68450.0	gap	= 0.011	
State-Based	9470	10240	38095	3600.13	0.02507	0.0001169	0.00905
Break based	26409	9649	4090	3600.37	0.01131	0.008108	0.01716
SCIP+	11384	725	7141	3600.1	0.008428	0.0005405	0.008993
SCIP+: col. gen.	11582	730	9729	3600.01	0.002972	0.005479	0.01046
Presolved break based	12912	9682	45530	1014.09	0.012	0.0	8.8e-05
la03_0_s_m	dual =	73250.0	opt =	73860.0	gap	= 0.0083	
State-Based	7703	8374	50583	3600.06	0.02899	0.001164	0.006615
Break based	17876	7972	32249	3600.69	0.008248	0.000704	0.003939
SCIP+	7480	645	2714	850.46	0.005282	0.0	0.0
SCIP+: col. gen.	7647	650	9383	1993.05	0.0006111	0.0	0.0
Presolved break based	8620	8006	149205	1298.04	0.0077	0.0	6.8e-05
la03_0_s_s	dual =	75790.0	opt =	76400.0	gap	= 0.0079	
State-Based	6055	6659	52402	2468.04	0.01954	0.0	2.618e-05
Break based	11248	6408	30453	3600.38	0.007924	0.001309	0.005217
SCIP+	4959	570	600	170.05	0.006368	0.0	0.0
SCIP+: col. gen.	5223	575	2340	521.24	0.001153	0.0	0.0
Presolved break based	5687	6435	54153	548.3	0.008	0.0	9.2e-05
la03_1_l_h	dual =	6995.0	opt =	6995.0	gap	= -0.0	
State-Based	9831	18149	1711	1568.01	0.0	0.0	0.0
Break based	23697	9856	1	2227.63	0.0	0.0	0.0
SCIP+	6002	735	697	1098.9	0.0	0.0	0.0
SCIP+: col. gen.	6796	870	159	238.62	0.0	0.0	0.0
Presolved break based	5980	9883	115	437.88	0.0	0.0	0.0
la03_1_l_l	dual =	6995.0	opt =	6995.0	gap	= -0.0	
State-Based	8181	15449	3531	1303.11	0.0	0.0	0.0
Break based	16488	8287	90	3600.22	0.0	0.002573	0.002573
SCIP+	5027	660	21	130.61	0.0	0.0	0.0
SCIP+: col. gen.	5803	795	202	245.96	0.0	0.0	0.0
Presolved break based	5017	8320	1	90.32	0.0	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la03_1_l_m	dual =	6995.0	opt =	6995.0	gap	= -0.0	
State-Based	6421	12569	950	194.27	0.0	0.0	0.0
Break based	10049	6607	1909	2364.25	0.0	0.0	0.0
SCIP+	3987	580	42	172.6	0.0	0.0	0.0
SCIP+: col. gen.	4571	715	99	172.32	0.0	0.0	0.0
Presolved break based	3974	6636	1	48.14	0.0	0.0	0.0
la03_1_l_s	dual =	∞	opt =	-∞	gap	= 0.0	
State-Based	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SCIP+	3159	685	0.0	0.08	0.0	0.0	0.0
SCIP+: col. gen.	3222	640	1	0.21	0.0	0.0	0.0
Presolved break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
la03_1_m_h	dual =	4351.0	opt =	4351.0	gap	= -0.0	
State-Based	10425	15501	3753	1275.91	0.0	0.0	0.0
Break based	28791	10465	1	3600.18	0.0	0.04964	0.04964
SCIP+	9182	765	111	910.32	0.0	0.0	0.0
SCIP+: col. gen.	7500	840	160	359.53	0.0	0.0	0.0
Presolved break based	12052	10503	12	1406.63	0.0	0.0	0.0
la03_1_m_l	dual =	4351.0	opt =	4351.0	gap	= 0.0	
State-Based	8775	13251	3413	1509.57	0.0	0.0	0.0
Break based	20703	8917	1	3603.78	0.0	0.01931	0.01931
SCIP+	5870	690	46	200.71	0.0	0.0	0.0
SCIP+: col. gen.	6271	765	263	383.64	0.0	0.0	0.0
Presolved break based	7707	8954	1	363.7	0.0	0.0	0.0
la03_1_m_m	dual =	4351.0	opt =	4351.0	gap	= -0.0	
State-Based	7015	10851	4752	1142.83	0.0	0.0	0.0
Break based	13287	7237	1	1351.45	0.0	0.0	0.0
SCIP+	4377	610	31	82.53	0.0	0.0	0.0
SCIP+: col. gen.	4838	685	70	84.76	0.0	0.0	0.0
Presolved break based	4696	7266	174	388.47	0.0	0.0	0.0
la03_1_m_s	dual =	4351.0	opt =	4351.0	gap	= -0.0	
State-Based	5380	8616	1560	287.41	0.0	0.0	0.0
Break based	7511	5653	1	707.76	0.0	0.0	0.0
SCIP+	3400	535	149	203.07	0.0	0.0	0.0
SCIP+: col. gen.	3970	610	81	132.05	0.0	0.0	0.0
Presolved break based	3383	5683	1	34.3	0.0	0.0	0.0
la03_1_r_h	dual =	5509.0	opt =	5509.0	gap	= -0.0	
State-Based	10880	13655	3925	1848.81	0.0	0.0	0.0
Break based	35192	10963	1	3600.49	0.0	0.001634	0.001634
SCIP+	11931	784	154	787.09	0.0	0.0	0.0
SCIP+: col. gen.	7233	821	155	299.76	0.0	0.0	0.0
Presolved break based	14653	10993	115	735.99	0.0	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la03_1_r_l	dual =	3901.0	opt =	3901.0	gap	= -0.0	
State-Based	8203	12378	3121	1187.45	0.0	0.0	0.0
Break based	20741	8384	1	3608.54	0.0	0.0123	0.0123
SCIP+	7311	692	86	391.46	0.0	0.0	0.0
SCIP+: col. gen.	5715	763	65	147.73	0.0	0.0	0.0
Presolved break based	8515	8417	1	360.76	0.0	0.0	0.0
la03_1_r_m	dual =	5368.0	opt =	5368.0	gap	= -0.0	
State-Based	6590	11238	39	159.24	0.0	0.0	0.0
Break based	12435	6747	1	2111.28	0.0	0.0	0.0
SCIP+	6746	597	83	183.38	0.0	0.0	0.0
SCIP+: col. gen.	5915	698	920	386.26	0.0	0.0	0.0
Presolved break based	7606	6782	1	238.66	0.0	0.0	0.0
la03_1_r_s	dual =	3844.0	opt =	3858.0	gap	= 0.0036	
State-Based	5322	7872	26681	3476.09	0.003629	0.0	0.0
Break based	8593	5547	3751	2256.81	0.003629	0.0	0.0
SCIP+	2830	500	43	103.11	0.003628	0.0	0.0
SCIP+: col. gen.	4159	604	165	186.63	0.0	0.0	0.0
Presolved break based	3344	5576	2379	201.41	0.0036	0.0	0.0
la03_1_s_h	dual =	1826.0	opt =	1826.0	gap	= 0.0	
State-Based	11117	11950	4375	1519.06	0.0	0.0	0.0
Break based	37822	11198	136	3600.18	0.0	0.1763	0.1763
SCIP+	27314	800	34	1279.58	0.0	0.0	0.0
SCIP+: col. gen.	11084	805	685	2053.84	0.0	0.0	0.0
Presolved break based	33403	11236	1	824.75	0.0	0.0	0.0
la03_1_s_l	dual =	1826.0	opt =	1826.0	gap	= -0.0	
State-Based	9467	10225	1637	331.49	0.0	0.0	0.0
Break based	26300	9649	58	3609.93	0.0	0.0575	0.0575
SCIP+	18959	725	49	486.65	0.0	0.0	0.0
SCIP+: col. gen.	7459	730	175	512.93	0.0	0.0	0.0
Presolved break based	22340	9688	404	2479.28	0.0	0.0	0.0
la03_1_s_m	dual =	1826.0	opt =	1826.0	gap	= -0.0	
State-Based	7707	8385	4079	654.26	0.0	0.0	0.0
Break based	17767	7972	115	3601.8	0.0	0.04545	0.04545
SCIP+	11287	645	77	228.41	0.0	0.0	0.0
SCIP+: col. gen.	5807	650	82	122.52	0.0	0.0	0.0
Presolved break based	14135	8008	1	311.59	0.0	0.0	0.0
la03_1_s_s	dual =	1826.0	opt =	1826.0	gap	= 0.0	
State-Based	6057	6660	1695	153.0	0.0	0.0	0.0
Break based	10927	6397	115	2489.12	0.0	0.0	0.0
SCIP+	5384	570	24	70.69	0.0	0.0	0.0
SCIP+: col. gen.	4450	575	66	92.81	0.0	0.0	0.0
Presolved break based	7721	6432	1	107.0	0.0	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la03_7_l_h	dual =	286100.0	opt =	289900.0	gap	= 0.013	
State-Based	9829	18148	17340	3600.05	0.03379	0.0002863	0.0235
Break based	23999	9857	1298	3600.45	0.01286	0.006683	0.01814
SCIP+	8144	735	16861	3600.11	0.01112	0.0001276	0.004261
SCIP+: col. gen.	8910	870	18436	3600.01	0.000231	0.0	0.001422
Presolved break based	9604	9885	92569	3600.11	0.012	0.00029	0.0053
la03_7_l_l	dual =	288000.0	opt =	289900.0	gap	= 0.0067	
State-Based	8182	15502	29618	3610.74	0.01112	0.001283	0.007052
Break based	16775	8288	23818	3600.59	0.006503	0.0	0.003024
SCIP+	6090	660	189	141.6	0.005242	0.0	0.0
SCIP+: col. gen.	5907	795	850	276.16	0.0004332	0.0	0.0
Presolved break based	6980	8319	216793	2174.18	0.0059	0.0	0.0
la03_7_l_m	dual =	288600.0	opt =	290100.0	gap	= 0.0052	
State-Based	6419	12568	29531	3600.09	0.01006	0.000686	0.005316
Break based	10333	6611	52308	3600.42	0.005245	0.0	0.0003586
SCIP+	4143	580	217	68.54	0.004335	0.0	0.0
SCIP+: col. gen.	4506	715	2012	334.91	0.0004779	0.0	0.0
Presolved break based	4958	6644	69578	592.17	0.0052	0.0	0.0
la03_7_l_s	dual =	∞	opt =	-∞	gap	= 0.0	
State-Based	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SCIP+	3159	685	0.0	0.04	0.0	0.0	0.0
SCIP+: col. gen.	3222	640	1	0.2	0.0	0.0	0.0
Presolved break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
la03_7_m_h	dual =	179300.0	opt =	181800.0	gap	= 0.014	
State-Based	10423	15500	12963	3600.03	0.03278	0.002871	0.02675
Break based	28970	10465	3106	3600.55	0.01367	0.03695	0.05037
SCIP+	11293	765	10727	3600.18	0.01056	0.0	0.00413
SCIP+: col. gen.	11279	840	7742	3600.02	0.000414	0.000187	0.004916
Presolved break based	13556	10501	146379	3600.21	0.014	0.00023	0.0053
la03_7_m_l	dual =	179700.0	opt =	181800.0	gap	= 0.012	
State-Based	8774	13262	29064	3601.95	0.01611	0.002398	0.01239
Break based	20882	8917	2874	3600.4	0.01183	0.00209	0.01295
SCIP+	7677	690	10320	2708.18	0.008834	0.0	0.0
SCIP+: col. gen.	8703	765	10522	3600.01	0.0003054	0.0	0.001727
Presolved break based	9217	8953	93285	3600.27	0.012	0.0	0.0049
la03_7_m_m	dual =	180100.0	opt =	181800.0	gap	= 0.0094	
State-Based	7013	10850	29061	3601.59	0.01559	0.00022	0.009106
Break based	13466	7237	32025	3601.52	0.00925	0.001155	0.006553
SCIP+	5290	610	16056	3600.05	0.007983	0.0	0.0005059
SCIP+: col. gen.	6192	685	8671	2281.99	0.0006144	0.0	0.0
Presolved break based	6008	7272	223574	3600.22	0.0092	0.0	0.0014

instance	vars	cons	nodes	time	relative root	relative primal	gap
la03_7_m_s	dual =	181600.0	opt =	182700.0	gap	= 0.0063	
State-Based	5378	8615	94425	3600.05	0.01008	0.0	0.001644
Break based	7820	5668	12551	575.65	0.006142	0.0	0.0
SCIP+	3484	535	4286	829.48	0.005888	0.0	0.0
SCIP+: col. gen.	4036	610	4894	998.16	0.0008633	0.0	0.0
Presolved break based	4074	5688	7025	86.79	0.006	0.0	0.0
la03_7_r_h	dual =	228200.0	opt =	229900.0	gap	= 0.0075	
State-Based	10181	16385	29131	3600.28	0.01985	0.002853	0.01205
Break based	27336	10232	7272	3600.52	0.007464	0.000274	0.005008
SCIP+	11711	754	647	275.87	0.006556	0.0	0.0
SCIP+: col. gen.	8969	851	2883	623.32	0.0001665	0.0	0.0
Presolved break based	14032	10255	100079	1429.26	0.0074	0.0	0.0
la03_7_r_l	dual =	252000.0	opt =	253900.0	gap	= 0.0076	
State-Based	8466	13651	34494	3600.1	0.00939	0.0	0.004923
Break based	20137	8558	26987	3600.19	0.007472	0.001197	0.005406
SCIP+	9570	682	7630	1068.97	0.007426	0.0	0.0
SCIP+: col. gen.	7751	773	10214	1779.93	0.0001673	0.0	0.0
Presolved break based	10973	8590	534535	3396.45	0.0074	0.0	0.0
la03_7_r_m	dual =	182600.0	opt =	184000.0	gap	= 0.0077	
State-Based	7451	8736	94936	3600.09	0.01237	0.0	0.002315
Break based	17007	7701	8024	3600.27	0.007534	0.001717	0.005415
SCIP+	7361	639	1629	475.82	0.006811	0.0	0.0
SCIP+: col. gen.	6656	657	1913	380.34	0.0006216	0.0	0.0
Presolved break based	8426	7736	18110	322.87	0.0073	0.0	0.0
la03_7_r_s	dual =	193800.0	opt =	195400.0	gap	= 0.0084	
State-Based	5135	8462	41628	3172.4	0.01262	0.0	0.0
Break based	7643	5255	14100	3341.16	0.008495	0.0	0.0
SCIP+	3499	532	1633	463.54	0.007511	0.0	0.0
SCIP+: col. gen.	4312	613	4028	1149.1	0.0005962	0.0	0.0
Presolved break based	4173	5276	25903	267.99	0.0087	0.0	0.0
la03_7_s_h	dual =	74800.0	opt =	76110.0	gap	= 0.017	
State-Based	11115	11949	26207	3600.01	0.02519	0.006477	0.0227
Break based	37920	11198	199	3600.86	0.01718	0.03525	0.05309
SCIP+	20877	800	2703	3600.14	0.01344	0.0006832	0.006468
SCIP+: col. gen.	15118	805	6654	3600.02	0.0004266	0.0005912	0.007338
Presolved break based	25202	11230	64605	3600.21	0.017	0.0045	0.015
la03_7_s_l	dual =	75320.0	opt =	76130.0	gap	= 0.011	
State-Based	9465	10224	29829	3600.06	0.0131	0.008183	0.01642
Break based	26409	9649	143	3600.57	0.01072	0.007999	0.01837
SCIP+	14576	725	3819	2052.36	0.0112	0.0	0.0
SCIP+: col. gen.	10783	730	4107	1975.47	0.0	0.0	0.0
Presolved break based	16365	9684	110596	3600.18	0.011	0.0	0.0012

instance	vars	cons	nodes	time	relative root	relative primal	gap
la03_7_s_m	dual =	75330.0	opt =	76130.0	gap	= 0.011	
State-Based	7705	8384	35337	3600.08	0.01302	0.00176	0.008623
Break based	17876	7972	3482	3600.08	0.01056	0.01135	0.02156
SCIP+	9487	645	1278	693.57	0.01088	0.0	0.0
SCIP+: col. gen.	7934	650	4242	1622.39	0.0	0.0	0.0
Presolved break based	10752	8007	169872	2343.72	0.011	0.0	1.3e-05
la03_7_s_s	dual =	75610.0	opt =	76130.0	gap	= 0.0069	
State-Based	6055	6659	67328	2275.34	0.00974	0.0	0.0
Break based	11248	6408	11001	1842.36	0.00687	0.0	0.0
SCIP+	5602	570	672	266.89	0.007544	0.0	0.0
SCIP+: col. gen.	5107	575	1266	368.14	0.0	0.0	0.0
Presolved break based	6505	6434	10432	176.53	0.0072	0.0	0.0
la03_8_l_h	dual =	451500.0	opt =	470000.0	gap	= 0.039	
State-Based	9784	18148	26408	3600.02	0.06782	0.0	0.04489
Break based	23779	9856	3019	3600.57	0.03943	0.0	0.03715
SCIP+	6207	735	11514	3600.32	0.03481	0.0	0.005849
SCIP+: col. gen.	8528	870	6355	2947.01	0.001898	0.0	0.0
Presolved break based	6517	9889	13986	530.2	0.034	0.0	0.0
la03_8_l_l	dual =	453800.0	opt =	470000.0	gap	= 0.035	
State-Based	8139	15538	29787	3603.07	0.0548	0.0006595	0.03515
Break based	16495	8287	5751	3600.2	0.03448	0.0	0.0146
SCIP+	5071	660	5113	1458.53	0.03036	0.0	0.0
SCIP+: col. gen.	6814	795	1852	1025.37	0.002341	0.0	0.0
Presolved break based	5278	8327	7482	438.23	0.031	0.0	0.0
la03_8_l_m	dual =	454900.0	opt =	470000.0	gap	= 0.032	
State-Based	6374	12568	19980	1205.12	0.05248	0.0	0.0
Break based	10036	6610	14713	3600.15	0.03215	0.0006595	0.01246
SCIP+	3920	580	9683	2379.36	0.02988	0.0	0.0
SCIP+: col. gen.	4682	715	2667	845.18	0.00206	0.0	0.0
Presolved break based	4071	6642	8890	326.37	0.029	0.0	0.0
la03_8_l_s	dual =	∞	opt =	$-\infty$	gap	= 0.0	
State-Based	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SCIP+	3159	685	0.0	0.03	0.0	0.0	0.0
SCIP+: col. gen.	3259	640	1	0.2	0.0	0.0	0.0
Presolved break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
la03_8_m_h	dual =	262300.0	opt =	268900.0	gap	= 0.024	
State-Based	10378	15500	25879	3600.02	0.0488	0.0001302	0.007988
Break based	28765	10465	160	3600.64	0.02456	0.04236	0.06579
SCIP+	7745	765	585	277.91	0.01033	0.0	0.0
SCIP+: col. gen.	8544	840	1871	876.24	0.004276	0.0	0.0
Presolved break based	8400	10502	3484	404.36	0.022	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la03_8_m_l	dual =	264500.0	opt =	268900.0	gap	= 0.016	
State-Based	8733	13310	15094	1396.82	0.04224	0.0	0.0
Break based	20587	8917	2282	3312.47	0.01633	0.0	0.0
SCIP+	5972	690	352	169.24	0.008191	0.0	0.0
SCIP+: col. gen.	6838	765	1230	626.01	0.003678	0.0	0.0
Presolved break based	6521	8960	2565	383.61	0.017	0.0	0.0
la03_8_m_m	dual =	265500.0	opt =	268900.0	gap	= 0.012	
State-Based	6968	10850	17573	934.45	0.04101	0.0	0.0
Break based	13030	7237	5325	3106.4	0.01252	0.0	0.0
SCIP+	4516	610	215	109.88	0.006509	0.0	0.0
SCIP+: col. gen.	5228	685	994	355.84	0.004145	0.0	0.0
Presolved break based	4913	7279	7565	263.0	0.014	0.0	0.0
la03_8_m_s	dual =	268800.0	opt =	269900.0	gap	= 0.0041	
State-Based	5333	8615	3598	172.96	0.02147	0.0	0.0
Break based	7551	5668	251	620.0	0.004132	0.0	0.0
SCIP+	3397	534	431	105.77	0.006944	0.0	0.0
SCIP+: col. gen.	3824	610	1218	216.96	0.00286	0.0	0.0
Presolved break based	3524	5688	225	40.32	0.0043	0.0	0.0
la03_8_r_h	dual =	205400.0	opt =	215300.0	gap	= 0.046	
State-Based	10112	15110	27392	3600.02	0.07892	0.0	0.008829
Break based	28968	10187	407	3601.5	0.04611	0.0005063	0.04361
SCIP+	8777	765	313	205.48	0.01698	0.0	0.0
SCIP+: col. gen.	7455	840	487	265.44	0.02462	0.0	0.0
Presolved break based	9670	10222	14651	686.73	0.042	0.0	0.0
la03_8_r_l	dual =	325900.0	opt =	328400.0	gap	= 0.0075	
State-Based	8643	13027	20088	1024.95	0.03498	0.0	0.0
Break based	20874	8791	4445	1425.34	0.007519	0.0	0.0
SCIP+	7133	690	2124	342.75	0.0086	0.0	0.0
SCIP+: col. gen.	6447	765	1600	303.31	0.001684	0.0	0.0
Presolved break based	7596	8825	1722	62.93	0.0075	0.0	0.0
la03_8_r_m	dual =	228200.0	opt =	231700.0	gap	= 0.015	
State-Based	7056	10639	22833	1321.2	0.04338	0.0	0.0
Break based	13686	7447	50665	3600.72	0.01518	0.0	0.006569
SCIP+	5219	611	9186	3600.11	0.01451	0.0007294	0.005284
SCIP+: col. gen.	5572	684	15661	3600.01	0.0007525	0.0	0.001971
Presolved break based	5248	7479	52387	507.85	0.014	0.0	0.0
la03_8_r_s	dual =	∞	opt =	-∞	gap	= 0.0	
State-Based	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SCIP+	5731	662	0.0	0.09	0.0	0.0	0.0
SCIP+: col. gen.	3362	617	1	0.28	0.0	0.0	0.0
Presolved break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la03_8_s_h	dual =	99720.0	opt =	102200.0	gap	= 0.025	
State-Based	11072	11950	37245	3600.07	0.02655	0.0	0.005736
Break based	37500	11197	2445	3600.63	0.02452	0.0002739	0.02315
SCIP+	16785	800	4428	2688.79	0.02052	0.0	0.0
SCIP+: col. gen.	11149	805	6012	3476.19	0.003535	0.0	0.0
Presolved break based	17722	11232	55725	795.16	0.023	0.0	0.0
la03_8_s_l	dual =	100100.0	opt =	102200.0	gap	= 0.021	
State-Based	9422	10225	58655	2948.39	0.02437	0.0	0.0
Break based	25871	9649	15625	3600.22	0.02091	0.0	0.005409
SCIP+	12002	725	3376	1206.92	0.0	0.0	0.0
SCIP+: col. gen.	9066	730	4364	1978.28	0.003525	0.0	0.0
Presolved break based	11857	9685	23741	494.09	0.02	0.0	0.0
la03_8_s_m	dual =	100100.0	opt =	102200.0	gap	= 0.021	
State-Based	7662	8385	34733	1051.41	0.02517	0.0	0.0
Break based	16994	7972	15409	3600.28	0.02054	0.0	0.004175
SCIP+	8109	645	3982	1011.96	0.01901	0.0	0.0
SCIP+: col. gen.	7308	650	5336	2346.1	0.003555	0.0	0.0
Presolved break based	8734	8008	6492	351.61	0.019	0.0	0.0
la03_8_s_s	dual =	100300.0	opt =	102200.0	gap	= 0.019	
State-Based	6012	6660	14275	434.63	0.02245	0.0	0.0
Break based	10563	6408	17898	1708.03	0.01877	0.0	0.0
SCIP+	5029	570	8885	2518.1	0.01796	0.0	0.0
SCIP+: col. gen.	5118	575	3818	1264.37	0.004007	0.0	0.0
Presolved break based	5951	6435	10891	270.54	0.018	0.0	0.0
la04_0_m_h	dual =	132900.0	opt =	135100.0	gap	= 0.016	
State-Based	11266	16770	25307	3600.03	0.02885	0.0009624	0.01968
Break based	35902	11219	573	3600.26	0.01632	0.0	0.01536
SCIP+	15921	810	2704	3600.14	0.01196	0.0009624	0.005182
SCIP+: col. gen.	12709	885	3188	2047.67	0.001777	0.0	0.0
Presolved break based	21128	11255	199463	3600.23	0.016	0.0	0.0047
la04_0_m_l	dual =	149300.0	opt =	151700.0	gap	= 0.016	
State-Based	9501	14310	18468	3600.04	0.03539	0.000389	0.01878
Break based	24256	9557	104	3600.52	0.01532	0.01189	0.02615
SCIP+	10666	730	2571	3600.1	0.01225	0.0005407	0.006253
SCIP+: col. gen.	11581	805	5566	3600.02	0.0003808	0.0	0.003622
Presolved break based	13170	9591	112856	2504.28	0.016	0.0	0.0
la04_0_m_m	dual =	162800.0	opt =	164600.0	gap	= 0.011	
State-Based	7634	11796	29498	3600.22	0.04428	0.006952	0.03172
Break based	15700	7784	13736	3600.4	0.01039	0.0	0.005291
SCIP+	6136	645	6451	3600.08	0.009001	0.0	0.002475
SCIP+: col. gen.	7567	720	2169	946.22	0.0006545	0.0	0.0
Presolved break based	8107	7823	88925	1283.12	0.011	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la04_0_m_s	dual =	172500.0	opt =	174300.0	gap	= 0.01	
State-Based	5878	9400	29416	3603.0	0.02789	0.001073	0.01454
Break based	9197	6118	15938	3600.25	0.01037	0.0006941	0.005186
SCIP+	3627	565	1380	424.92	0.009242	0.0	0.0
SCIP+: col. gen.	4608	640	3606	1134.75	0.0003662	0.0	0.0
Presolved break based	4526	6144	114496	1185.37	0.011	0.0	1.7e-05
la04_0_r_h	dual =	131700.0	opt =	133300.0	gap	= 0.012	
State-Based	11137	17214	17553	3600.04	0.03241	0.0005553	0.01914
Break based	35401	11106	3234	3600.42	0.01153	0.005388	0.01595
SCIP+	10742	804	962	492.37	0.007423	0.0	0.0
SCIP+: col. gen.	9166	891	859	414.83	0.001675	0.0	0.0
Presolved break based	14873	11133	19953	394.93	0.011	0.0	0.0
la04_0_r_l	dual =	137800.0	opt =	140000.0	gap	= 0.016	
State-Based	9025	13786	29428	3600.16	0.04006	0.0	0.008261
Break based	24175	9155	795	3600.15	0.0154	4.286e-05	0.01443
SCIP+	10799	728	9634	3600.09	0.0131	0.0	0.007795
SCIP+: col. gen.	11143	807	17931	3600.02	0.0008536	0.0	0.002466
Presolved break based	12504	9191	13628	689.4	0.016	0.0	0.0
la04_0_r_m	dual =	158300.0	opt =	159900.0	gap	= 0.01	
State-Based	7610	11176	20862	3600.02	0.04172	0.0005502	0.01621
Break based	16548	7781	6549	3600.1	0.01001	0.001363	0.01007
SCIP+	9474	650	15734	3600.13	0.007896	0.0002188	0.005033
SCIP+: col. gen.	9940	716	13714	3600.01	0.001017	8.754e-05	0.003814
Presolved break based	10877	7818	78860	1786.46	0.0089	0.0	0.0
la04_0_r_s	dual =	201100.0	opt =	201400.0	gap	= 0.0014	
State-Based	5968	8262	5507	210.56	0.005665	0.0	0.0
Break based	10810	6213	1915	413.54	0.001522	0.0	0.0
SCIP+	4187	578	894	148.38	0.001559	0.0	0.0
SCIP+: col. gen.	4487	627	1045	121.74	0.000411	0.0	0.0
Presolved break based	4728	6229	1087	36.37	0.0013	0.0	0.0
la04_0_s_h	dual =	55760.0	opt =	56570.0	gap	= 0.014	
State-Based	11962	12869	29254	3600.1	0.02613	0.002634	0.01505
Break based	43227	11954	2982	3600.38	0.01424	0.002104	0.01561
SCIP+	23558	845	3235	3600.22	0.01222	0.002104	0.0145
SCIP+: col. gen.	16081	850	7488	3600.02	0.002522	0.0008486	0.006417
Presolved break based	28133	11991	217579	3457.1	0.014	0.0	5.3e-05
la04_0_s_l	dual =	62630.0	opt =	63000.0	gap	= 0.0059	
State-Based	10199	11014	29291	3600.03	0.01738	0.0	0.006936
Break based	30263	10283	4716	2793.64	0.005969	0.0	0.0
SCIP+	17290	765	714	901.53	0.004799	0.0	0.0
SCIP+: col. gen.	12023	770	2082	1577.84	0.0005197	0.0	0.0
Presolved break based	19115	10317	1234	426.76	0.0057	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la04_0_s_m	dual =	68570.0	opt =	69120.0	gap	= 0.008	
State-Based	8329	9059	36093	3600.13	0.0256	0.0	0.002902
Break based	20497	8519	10456	3329.69	0.00793	0.0	0.0
SCIP+	11085	680	1035	576.37	0.006983	0.0	0.0
SCIP+: col. gen.	10320	685	6334	2498.33	0.001757	0.0	0.0
Presolved break based	12722	8555	8092	384.38	0.0088	0.0	0.0
la04_0_s_s	dual =	74190.0	opt =	74620.0	gap	= 0.0058	
State-Based	6573	7239	30473	3600.03	0.02228	0.0	0.005579
Break based	12795	6847	22184	2577.68	0.005805	0.0	0.0
SCIP+	6081	600	329	103.25	0.004853	0.0	0.0
SCIP+: col. gen.	6421	605	1214	371.14	0.0002634	0.0	0.0
Presolved break based	7226	6880	89627	865.37	0.0057	0.0	0.0
la04_1_l_h	dual =	6299.0	opt =	6317.0	gap	= 0.0028	
State-Based	10661	19620	3609	3600.07	0.002849	0.0009498	0.00381
Break based	27949	10595	1	3600.22	0.002849	0.008548	0.01143
SCIP+	6451	780	290	3600.12	0.002849	0.005699	0.008573
SCIP+: col. gen.	8682	915	1134	3600.01	0.0	0.003799	0.006668
Presolved break based	6414	10623	38265	3600.3	0.0028	0.00047	0.0033
la04_1_l_l	dual =	6299.0	opt =	6317.0	gap	= 0.0028	
State-Based	8901	16740	5155	3600.02	0.002849	0.003324	0.006191
Break based	19533	8939	1	3600.33	0.002849	0.005699	0.008573
SCIP+	5411	700	687	3600.07	0.002849	0.005699	0.008525
SCIP+: col. gen.	6939	835	1433	3600.01	0.0	0.002375	0.005239
Presolved break based	5398	8967	117849	3600.08	0.0028	0.00047	0.0033
la04_1_l_m	dual =	6299.0	opt =	6317.0	gap	= 0.0028	
State-Based	7031	13680	7284	3600.03	0.002849	0.0009498	0.00381
Break based	11971	7154	1230	3600.44	0.002849	0.0019	0.004763
SCIP+	4306	615	2193	3600.09	0.002849	0.0004749	0.002213
SCIP+: col. gen.	5467	750	3271	3600.02	0.0	0.0	0.002858
Presolved break based	4284	7178	82063	3600.22	0.0028	0.0	0.0029
la04_1_l_s	dual =	6299.0	opt =	6335.0	gap	= 0.0057	
State-Based	5329	10858	10886	3600.01	0.005683	0.0009471	0.006668
Break based	6195	5459	8724	3426.64	0.005683	0.0	0.0
SCIP+	3219	535	1964	1121.67	0.005683	0.0	0.0
SCIP+: col. gen.	4174	670	7738	3600.01	0.0	0.0	0.001754
Presolved break based	3258	5498	13463	925.53	0.0057	0.0	0.0
la04_1_m_h	dual =	3872.0	opt =	3890.0	gap	= 0.0046	
State-Based	11261	16710	5968	3600.04	0.004627	0.001542	0.006198
Break based	35798	11219	163	3600.11	0.004627	0.01722	0.02195
SCIP+	11802	810	246	3600.15	0.004627	0.01337	0.01808
SCIP+: col. gen.	9959	885	1864	3600.02	0.0	0.0	0.004649
Presolved break based	17165	11258	15527	3600.18	0.0046	0.0015	0.0062

instance	vars	cons	nodes	time	relative root	relative primal	gap
la04_1_m_l	dual =	3872.0	opt =	3890.0	gap	= 0.0046	
State-Based	9501	14310	5051	3600.04	0.004627	0.0	0.004649
Break based	24126	9557	1	3605.01	0.004627	0.0383	0.04313
SCIP+	7330	730	477	3600.09	0.004627	0.00437	0.009039
SCIP+: col. gen.	8255	805	1281	3600.02	0.0	0.00617	0.01085
Presolved break based	9518	9592	57357	3600.33	0.0046	0.0015	0.0062
la04_1_m_m	dual =	3872.0	opt =	3890.0	gap	= 0.0046	
State-Based	7631	11760	8355	3600.01	0.004627	0.0007712	0.005424
Break based	15553	7784	199	3602.05	0.004627	0.0005141	0.005165
SCIP+	4748	645	1619	3600.08	0.004627	0.0002571	0.003599
SCIP+: col. gen.	6684	720	3881	3600.01	0.0	0.0	0.004649
Presolved break based	5448	7815	102537	3600.18	0.0046	0.00026	0.0049
la04_1_m_s	dual =	3872.0	opt =	3890.0	gap	= 0.0046	
State-Based	5875	9364	9061	3600.05	0.004627	0.001285	0.00594
Break based	8821	6102	3029	3600.19	0.004627	0.00437	0.009039
SCIP+	3656	565	2653	1509.32	0.004627	0.0	0.0
SCIP+: col. gen.	4655	640	8385	3600.03	0.0	0.0	0.001325
Presolved break based	3650	6134	90400	3600.07	0.0046	0.00077	0.0054
la04_1_r_h	dual =	3740.0	opt =	3756.0	gap	= 0.0043	
State-Based	11059	17125	3759	3600.04	0.00426	0.002929	0.007219
Break based	32924	10998	1	3600.26	0.00426	0.03887	0.04332
SCIP+	17220	803	232	3600.13	0.00426	0.0002662	0.004545
SCIP+: col. gen.	11179	892	1063	3600.02	0.0	0.004792	0.009091
Presolved break based	19109	11028	4831	3600.12	0.0043	0.0056	0.0099
la04_1_r_l	dual =	4476.0	opt =	4483.0	gap	= 0.0016	
State-Based	9173	14822	6324	3600.03	0.001561	0.0008923	0.002458
Break based	23276	9229	115	3601.27	0.001561	0.005354	0.006926
SCIP+	5599	720	729	3600.11	0.001561	0.008253	0.009786
SCIP+: col. gen.	8015	815	3819	3600.01	0.0	0.0	0.001564
Presolved break based	6312	9258	48061	3600.15	0.0016	0.0	0.0016
la04_1_r_m	dual =	2894.0	opt =	2903.0	gap	= 0.0031	
State-Based	8215	9422	16269	3600.02	0.0031	0.002067	0.005183
Break based	19559	8380	1	3600.08	0.0031	0.01343	0.01659
SCIP+	8373	675	573	3600.07	0.0031	0.003789	0.00681
SCIP+: col. gen.	8405	690	1425	3600.04	0.0	0.004823	0.007947
Presolved break based	9381	8419	118756	3600.11	0.0031	0.0014	0.0045
la04_1_r_s	dual =	3525.0	opt =	3561.0	gap	= 0.01	
State-Based	5507	9088	9862	3600.04	0.01011	0.00337	0.01333
Break based	8570	5591	277	3600.26	0.01011	0.00337	0.01362
SCIP+	5101	561	955	1676.01	0.01011	0.0	0.0
SCIP+: col. gen.	5932	645	3099	3600.01	0.0	0.0008425	0.004626
Presolved break based	5995	5629	132524	3600.23	0.01	0.0	0.0025

instance	vars	cons	nodes	time	relative root	relative primal	gap
la04_1_s_h	dual =	1695.0	opt =	1702.0	gap	= 0.0041	
State-Based	11961	12855	4875	3600.08	0.004113	0.009401	0.01357
Break based	43132	11954	1	3610.18	0.004113	0.1016	0.1062
SCIP+	25084	845	12	3600.16	0.004113	0.0188	0.02301
SCIP+: col. gen.	12066	850	971	3600.02	0.0	0.007638	0.0118
Presolved break based	30833	11994	5723	3600.09	0.0041	0.0035	0.0077
la04_1_s_l	dual =	1695.0	opt =	1702.0	gap	= 0.0041	
State-Based	10201	11015	5993	3600.06	0.004113	0.0005875	0.00472
Break based	30158	10283	12	3602.66	0.004113	0.04289	0.0472
SCIP+	16532	765	625	3600.11	0.004113	0.0047	0.00885
SCIP+: col. gen.	10140	770	1022	3600.02	0.0	0.008813	0.01298
Presolved break based	19356	10323	25539	3600.07	0.0041	0.00059	0.0047
la04_1_s_m	dual =	1695.0	opt =	1702.0	gap	= 0.0041	
State-Based	8331	9060	10235	3600.02	0.004113	0.001175	0.00531
Break based	20404	8519	171	3600.87	0.004113	0.00235	0.00649
SCIP+	9654	680	1395	3600.07	0.004113	0.0	0.00413
SCIP+: col. gen.	7859	685	1060	3600.03	0.0	0.0047	0.00885
Presolved break based	11538	8558	77106	3600.28	0.0041	0.00059	0.0047
la04_1_s_s	dual =	1695.0	opt =	1702.0	gap	= 0.0041	
State-Based	6571	7220	9733	3600.06	0.004113	0.0005875	0.00472
Break based	12546	6840	200	3600.33	0.004113	0.003525	0.00767
SCIP+	4962	600	3010	3600.06	0.004113	0.0	0.001803
SCIP+: col. gen.	6430	605	2716	3600.01	0.0	0.0	0.00413
Presolved break based	5920	6878	110778	3600.78	0.0041	0.00059	0.0047
la04_7_l_h	dual =	251100.0	opt =	253000.0	gap	= 0.0076	
State-Based	10663	19656	7255	3600.03	0.03281	0.003233	0.0298
Break based	28184	10595	100	3602.21	0.007521	0.02695	0.03396
SCIP+	7136	780	1010	720.19	0.006718	0.0	0.0
SCIP+: col. gen.	8275	915	1052	1011.26	0.0	0.0	0.0
Presolved break based	9079	10621	120319	3469.82	0.0076	0.0	0.0
la04_7_l_l	dual =	254200.0	opt =	256500.0	gap	= 0.009	
State-Based	8901	16740	16897	3600.02	0.02469	0.002581	0.01923
Break based	19768	8939	17260	3600.31	0.009053	0.001832	0.006845
SCIP+	5339	700	5781	2661.36	0.008068	0.0	0.0
SCIP+: col. gen.	7377	835	7602	3600.01	0.0001643	0.0005146	0.004204
Presolved break based	6073	8971	127979	3600.14	0.0087	0.0	0.003
la04_7_l_m	dual =	254300.0	opt =	256500.0	gap	= 0.0086	
State-Based	7031	13680	29854	3600.14	0.01697	0.001587	0.01313
Break based	12203	7154	33355	3600.48	0.008826	0.001228	0.005021
SCIP+	4258	615	1713	663.68	0.007329	0.0	0.0
SCIP+: col. gen.	5539	750	10315	3600.01	0.000162	0.0	0.00123
Presolved break based	4246	7182	361631	3600.2	0.0084	0.0	0.0022

instance	vars	cons	nodes	time	relative root	relative primal	gap
la04_7_l_s	dual =	256600.0	opt =	258200.0	gap	= 0.0062	
State-Based	5331	10894	68013	3208.97	0.01323	0.0	0.0
Break based	6470	5470	11822	1265.18	0.006272	0.0	0.0
SCIP+	3209	535	3856	969.02	0.005935	0.0	0.0
SCIP+: col. gen.	3813	670	4311	922.33	0.0004228	0.0	0.0
Presolved break based	3256	5501	12930	268.17	0.0063	0.0	0.0
la04_7_m_h	dual =	156800.0	opt =	159900.0	gap	= 0.019	
State-Based	11261	16710	5730	3600.05	0.03715	0.01354	0.04505
Break based	35963	11219	356	3608.78	0.01899	0.005511	0.02472
SCIP+	16734	810	3820	3600.15	0.01445	0.0007819	0.004537
SCIP+: col. gen.	12381	885	4136	3600.02	0.0	0.0002565	0.004907
Presolved break based	22201	11254	49377	3600.2	0.019	0.0019	0.016
la04_7_m_l	dual =	159000.0	opt =	161600.0	gap	= 0.016	
State-Based	9501	14310	18886	3600.03	0.02752	0.00156	0.02081
Break based	24301	9557	3679	3600.27	0.01603	0.02966	0.04566
SCIP+	10339	730	560	694.03	0.01429	0.0	0.0
SCIP+: col. gen.	9960	805	4364	3600.01	0.0	0.005577	0.01286
Presolved break based	13171	9596	54577	3605.35	0.016	0.0	0.0081
la04_7_m_m	dual =	159200.0	opt =	161600.0	gap	= 0.015	
State-Based	7631	11760	29105	3607.56	0.02451	0.0051	0.02526
Break based	15728	7784	5381	3600.3	0.01441	0.004512	0.01751
SCIP+	5735	645	790	587.57	0.01247	0.0	0.0
SCIP+: col. gen.	6941	720	6545	3547.12	0.0003155	0.0	0.0
Presolved break based	7799	7822	117552	3600.13	0.014	0.0	0.003
la04_7_m_s	dual =	160000.0	opt =	162600.0	gap	= 0.016	
State-Based	5876	9376	29329	3600.44	0.02551	0.001193	0.0177
Break based	9197	6118	21964	3600.53	0.0162	0.0007134	0.009341
SCIP+	3623	565	7090	2337.11	0.0142	0.0	0.0
SCIP+: col. gen.	4683	640	9571	3600.01	0.0002455	0.0	0.002241
Presolved break based	4226	6146	134336	2431.24	0.016	0.0	0.0
la04_7_r_h	dual =	149300.0	opt =	151100.0	gap	= 0.012	
State-Based	11297	16481	11792	3600.04	0.02541	0.01577	0.03692
Break based	34159	11275	1	3608.43	0.01175	0.0239	0.03607
SCIP+	25180	812	1030	3600.25	0.01098	0.003045	0.009018
SCIP+: col. gen.	14803	883	3868	3600.01	0.0	0.0	0.004301
Presolved break based	29339	11314	32418	3600.26	0.012	0.0	0.005
la04_7_r_l	dual =	137200.0	opt =	138000.0	gap	= 0.0059	
State-Based	9127	14011	23339	3600.13	0.01536	0.001522	0.005134
Break based	24401	9124	3141	3600.54	0.005983	0.001848	0.006683
SCIP+	8749	729	99	156.65	0.003947	0.0	0.0
SCIP+: col. gen.	6672	806	257	183.87	0.0003712	0.0	0.0
Presolved break based	10562	9153	6943	566.96	0.0049	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la04_7_r_m	dual =	205100.0	opt =	206100.0	gap	= 0.005	
State-Based	7463	12311	26086	3600.05	0.01018	0.0	0.005257
Break based	15058	7586	5834	3600.4	0.005116	0.006282	0.01007
SCIP+	8080	635	970	401.63	0.004017	0.0	0.0
SCIP+: col. gen.	6151	730	3549	1076.15	0.0001964	0.0	0.0
Presolved break based	9745	7619	22484	543.26	0.0048	0.0	0.0
la04_7_r_s	dual =	165600.0	opt =	168000.0	gap	= 0.014	
State-Based	5343	10356	40908	3600.07	0.0219	0.0	0.004833
Break based	7001	5418	11011	2563.2	0.01401	0.0	0.0
SCIP+	3230	543	205	145.13	0.01234	0.0	0.0
SCIP+: col. gen.	3801	662	336	144.5	0.000358	0.0	0.0
Presolved break based	3229	5438	15433	412.2	0.013	0.0	0.0
la04_7_s_h	dual =	65230.0	opt =	66300.0	gap	= 0.016	
State-Based	11952	12837	11999	3600.03	0.03408	0.002338	0.0279
Break based	43227	11954	815	3600.78	0.01615	0.03226	0.04881
SCIP+	26442	845	1074	3600.31	0.01155	0.0007541	0.009993
SCIP+: col. gen.	16761	850	4244	3600.02	0.0	0.001961	0.008472
Presolved break based	31008	11987	56151	3600.37	0.016	0.0044	0.015
la04_7_s_l	dual =	65780.0	opt =	66630.0	gap	= 0.013	
State-Based	10193	11002	29505	3600.14	0.01783	0.009124	0.024
Break based	30263	10283	33	3602.23	0.01276	0.2031	0.2182
SCIP+	18736	765	5332	3600.13	0.01208	0.0005253	0.005525
SCIP+: col. gen.	13216	770	3932	3600.02	0.0	0.01868	0.02673
Presolved break based	20794	10319	56246	3600.27	0.013	0.00065	0.0069
la04_7_s_m	dual =	65820.0	opt =	66630.0	gap	= 0.012	
State-Based	8322	9042	29610	3600.05	0.01675	0.004007	0.01676
Break based	20509	8519	1996	3601.12	0.01224	0.02035	0.03167
SCIP+	11791	680	2638	1293.07	0.01167	0.0	0.0
SCIP+: col. gen.	10926	685	6012	3600.01	0.0001232	0.001096	0.007778
Presolved break based	13487	8557	101170	3600.2	0.012	0.0	0.003
la04_7_s_s	dual =	65950.0	opt =	66630.0	gap	= 0.01	
State-Based	6562	7202	35101	3600.06	0.01442	0.0007204	0.006612
Break based	12795	6847	8096	3600.35	0.01021	0.01417	0.02228
SCIP+	6646	600	834	407.43	0.009039	0.0	0.0
SCIP+: col. gen.	6320	605	2443	1280.33	0.0001363	0.0	0.0
Presolved break based	7946	6879	82712	991.17	0.01	0.0	0.0
la04_8_l_h	dual =	597200.0	opt =	615800.0	gap	= 0.03	
State-Based	10617	19657	28894	3600.03	0.05203	0.001554	0.03183
Break based	28077	10595	2260	3600.5	0.0303	0.001863	0.03006
SCIP+	9007	780	10002	3600.11	0.02755	0.0	0.009092
SCIP+: col. gen.	8632	915	7782	3600.02	0.0006208	0.0	0.009245
Presolved break based	10279	10624	67309	1024.39	0.03	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la04_8_l_l	dual =	598000.0	opt =	615800.0	gap	= 0.029	
State-Based	8854	16723	29406	3600.06	0.05048	0.0	0.02744
Break based	19600	8939	6238	3600.52	0.02901	0.0	0.02489
SCIP+	6573	700	9969	3600.07	0.02706	2.111e-05	0.003083
SCIP+: col. gen.	6770	835	9079	3600.02	0.0005593	0.0	0.003526
Presolved break based	7724	8975	282274	3145.89	0.029	0.0	5.4e-05
la04_8_l_m	dual =	602600.0	opt =	615900.0	gap	= 0.022	
State-Based	6984	13663	30042	3600.02	0.03815	0.0	0.02107
Break based	11919	7154	22504	3600.07	0.02171	0.0	0.00514
SCIP+	4401	615	4134	1082.92	0.01989	0.0	0.0
SCIP+: col. gen.	5198	750	3566	1157.1	0.001487	0.0	0.0
Presolved break based	5417	7194	113271	816.31	0.021	0.0	0.0
la04_8_l_s	dual =	618100.0	opt =	640500.0	gap	= 0.035	
State-Based	5286	10913	95049	3600.07	0.04892	0.0	0.005731
Break based	6337	5472	16890	3600.23	0.03503	0.002965	0.02582
SCIP+	3166	535	10712	3600.06	0.03459	0.0009321	0.005818
SCIP+: col. gen.	4110	670	11546	3600.01	0.002565	0.0001077	0.01354
Presolved break based	3576	5501	64351	516.66	0.035	0.0	0.0
la04_8_m_h	dual =	352600.0	opt =	363600.0	gap	= 0.03	
State-Based	11209	16672	23056	3600.02	0.05641	0.02062	0.06303
Break based	35841	11219	745	3600.36	0.03027	0.03262	0.06031
SCIP+	12021	810	5428	3600.13	0.02228	0.0	0.01185
SCIP+: col. gen.	10637	885	5187	3600.02	0.002647	0.0001017	0.01544
Presolved break based	15726	11253	77113	2861.83	0.029	0.0	0.0
la04_8_m_l	dual =	355200.0	opt =	363800.0	gap	= 0.024	
State-Based	9449	14272	29805	3600.12	0.05618	0.009061	0.03876
Break based	24085	9557	1454	3600.09	0.02358	0.008503	0.02893
SCIP+	8705	730	4658	3600.1	0.0229	0.01268	0.01862
SCIP+: col. gen.	9258	805	6678	3600.01	0.002734	0.005737	0.02092
Presolved break based	10094	9595	25077	831.51	0.026	0.0	0.0
la04_8_m_m	dual =	356200.0	opt =	364200.0	gap	= 0.022	
State-Based	7579	11722	37228	3600.07	0.04959	0.0	0.008958
Break based	15289	7784	16396	3049.57	0.02208	0.0	0.0
SCIP+	5923	645	9155	3404.13	0.02189	0.0	0.0
SCIP+: col. gen.	6531	720	7412	3103.97	0.002887	0.0	0.0
Presolved break based	7108	7821	18630	355.5	0.023	0.0	0.0
la04_8_m_s	dual =	358800.0	opt =	364200.0	gap	= 0.015	
State-Based	5826	9357	4839	536.73	0.03894	0.0	0.0
Break based	8891	6118	1761	3065.95	0.01498	0.0	0.0
SCIP+	3721	565	447	210.17	0.01475	0.0	0.0
SCIP+: col. gen.	4242	640	1069	422.29	0.004092	0.0	0.0
Presolved break based	4632	6141	5011	229.16	0.016	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la04_8_r_h	dual =	463000.0	opt =	479300.0	gap	= 0.034	
State-Based	11247	16126	29283	3600.16	0.04232	0.005209	0.03635
Break based	36774	11216	1260	3602.16	0.03401	0.009995	0.0433
SCIP+	13755	814	4853	3600.14	0.032	0.004997	0.03027
SCIP+: col. gen.	11002	881	8218	3600.03	0.001175	0.00562	0.02478
Presolved break based	16933	11246	201277	3250.77	0.035	0.0	0.0
la04_8_r_l	dual =	193700.0	opt =	194500.0	gap	= 0.004	
State-Based	9599	13164	4613	696.41	0.02476	0.0	0.0
Break based	26130	9627	264	2565.3	0.003882	0.0	0.0
SCIP+	11310	740	52	111.88	0.003428	0.0	0.0
SCIP+: col. gen.	6617	795	57	36.4	0.0006572	0.0	0.0
Presolved break based	12419	9664	444	203.0	0.0047	0.0	0.0
la04_8_r_m	dual =	433500.0	opt =	442600.0	gap	= 0.021	
State-Based	7011	12635	30277	3605.54	0.0609	0.001511	0.01352
Break based	13169	7166	13021	2817.74	0.02055	0.0	0.0
SCIP+	5974	625	4039	956.85	0.02123	0.0	0.0
SCIP+: col. gen.	5512	740	10734	2286.92	0.0006226	0.0	0.0
Presolved break based	7364	7204	37402	346.47	0.021	0.0	5.9e-05
la04_8_r_s	dual =	447600.0	opt =	458500.0	gap	= 0.024	
State-Based	5884	8449	69648	3600.06	0.02922	0.0	0.003063
Break based	9887	6086	19616	3600.26	0.02359	0.007698	0.01605
SCIP+	4220	577	5753	3600.05	0.02568	0.009403	0.01571
SCIP+: col. gen.	5856	628	6459	3600.02	0.002248	0.0	0.01201
Presolved break based	5196	6108	7155	179.81	0.026	0.0	0.0
la04_8_s_h	dual =	140600.0	opt =	144000.0	gap	= 0.023	
State-Based	11909	12838	29868	3600.12	0.02637	0.001799	0.01491
Break based	42964	11954	476	3600.51	0.02352	0.004667	0.02501
SCIP+	16956	845	6749	2541.84	0.0	0.0	0.0
SCIP+: col. gen.	13512	850	4994	3237.13	0.00202	0.0	0.0
Presolved break based	18824	11987	33670	1102.98	0.023	0.0	0.0
la04_8_s_l	dual =	141500.0	opt =	144000.0	gap	= 0.017	
State-Based	10150	11003	32878	3600.14	0.0258	0.0	0.009748
Break based	29861	10283	127	3601.67	0.01731	0.01511	0.03143
SCIP+	13083	765	5960	1981.21	0.01687	0.0	0.0
SCIP+: col. gen.	10816	770	5378	2399.23	0.001796	0.0	0.0
Presolved break based	13005	10318	11010	621.11	0.017	0.0	0.0
la04_8_s_m	dual =	141900.0	opt =	144200.0	gap	= 0.016	
State-Based	8279	9043	90338	3600.09	0.02257	0.001227	0.006776
Break based	19571	8519	7300	3521.2	0.01618	0.0	0.0
SCIP+	9015	680	6082	1487.38	0.01578	0.0	0.0
SCIP+: col. gen.	7559	685	1696	777.32	0.002192	0.0	0.0
Presolved break based	9332	8557	3620	296.22	0.017	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la04_8_s_s	dual =	142000.0	opt =	144200.0	gap	= 0.016	
State-Based	6520	7208	9548	375.49	0.01868	0.0	0.0
Break based	12026	6847	1725	1449.07	0.01571	0.0	0.0
SCIP+	5757	600	4580	1041.1	0.01468	0.0	0.0
SCIP+: col. gen.	5120	605	897	317.49	0.003236	0.0	0.0
Presolved break based	6355	6878	6204	125.61	0.015	0.0	0.0
la05_0_l_h	dual =	227000.0	opt =	228300.0	gap	= 0.0058	
State-Based	9133	16942	16695	3600.02	0.03199	0.0	0.01099
Break based	20958	9245	9629	3600.21	0.006055	0.0	0.001764
SCIP+	6161	700	11435	2590.82	0.005476	0.0	0.0
SCIP+: col. gen.	8404	835	15558	3600.01	0.0004857	0.0	0.001063
Presolved break based	7254	9281	3823	228.44	0.006	0.0	0.0
la05_0_l_l	dual =	242400.0	opt =	243700.0	gap	= 0.0051	
State-Based	7593	14422	16859	3600.04	0.03358	7.388e-05	0.01486
Break based	14581	7775	11372	3600.35	0.00522	0.0	0.001826
SCIP+	4713	630	191	94.63	0.004448	0.0	0.0
SCIP+: col. gen.	5947	765	574	205.89	0.0002447	0.0	0.0
Presolved break based	5132	7809	2500	108.4	0.0051	0.0	4.1e-05
la05_0_l_m	dual =	254900.0	opt =	256600.0	gap	= 0.0066	
State-Based	5955	11751	29375	3600.17	0.02995	0.0002299	0.01055
Break based	8967	6209	32669	3600.85	0.006466	0.0004638	0.003067
SCIP+	3659	555	2423	559.96	0.006108	0.0	0.0
SCIP+: col. gen.	4511	690	10843	957.91	0.0	0.0	0.0
Presolved break based	3719	6230	74696	673.1	0.0065	0.0	0.0
la05_0_l_s	dual =	∞	opt =	-∞	gap	= 0.0	
State-Based	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SCIP+	3184	665	0.0	0.04	0.0	0.0	0.0
SCIP+: col. gen.	2865	620	1	0.29	0.0	0.0	0.0
Presolved break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
la05_0_m_h	dual =	140000.0	opt =	141300.0	gap	= 0.0094	
State-Based	9733	14512	27504	3600.02	0.0289	0.0001061	0.008851
Break based	25810	9866	1963	3600.23	0.009201	0.0004387	0.008675
SCIP+	12506	730	6931	3600.1	0.00736	0.0	0.002914
SCIP+: col. gen.	11416	805	9604	3240.63	0.0003961	0.0	0.0
Presolved break based	14768	9900	7434	553.3	0.0091	0.0	0.0
la05_0_m_l	dual =	150900.0	opt =	152000.0	gap	= 0.0072	
State-Based	8193	12412	26389	3600.04	0.03255	0.002934	0.01177
Break based	18602	8405	5842	3600.45	0.007522	0.0	0.005397
SCIP+	8814	660	1324	1031.63	0.005776	0.0	0.0
SCIP+: col. gen.	8288	735	1718	928.92	0.0001623	0.0	0.0
Presolved break based	10481	8442	7978	622.99	0.0071	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la05_0_m_m	dual =	161100.0	opt =	162500.0	gap	= 0.0085	
State-Based	6545	10164	19131	3600.02	0.03873	0.004555	0.0306
Break based	12185	6842	9424	3600.43	0.00849	0.002493	0.009708
SCIP+	5245	585	1640	574.65	0.007817	0.0	0.0
SCIP+: col. gen.	6451	660	4615	1318.78	0.0	0.0	0.0
Presolved break based	6758	6870	246817	3600.55	0.0085	0.0	0.00094
la05_0_m_s	dual =	162700.0	opt =	163700.0	gap	= 0.0061	
State-Based	5039	8098	29982	3600.28	0.01751	0.0	0.004073
Break based	6938	5371	20732	3600.22	0.006096	0.0009347	0.003626
SCIP+	3471	515	6820	1561.75	0.00574	0.0	0.0
SCIP+: col. gen.	3995	590	2641	353.02	0.0001278	0.0	0.0
Presolved break based	3912	5392	42468	461.98	0.0062	0.0	0.0
la05_0_r_h	dual =	159800.0	opt =	161700.0	gap	= 0.012	
State-Based	9318	14104	29857	3603.82	0.05162	0.000167	0.01066
Break based	25870	9307	4130	3600.31	0.01143	0.003136	0.01291
SCIP+	12260	728	8151	2697.3	0.01035	0.0	0.0
SCIP+: col. gen.	12393	807	13226	3600.03	0.001534	0.0	0.002705
Presolved break based	14281	9344	82568	1431.68	0.011	0.0	0.0
la05_0_r_l	dual =	120400.0	opt =	121900.0	gap	= 0.012	
State-Based	7902	12205	29611	3600.17	0.0416	0.0004267	0.009497
Break based	18091	8061	220	3600.4	0.01221	0.03693	0.0488
SCIP+	10292	656	118	277.84	0.008328	0.0	0.0
SCIP+: col. gen.	9152	739	3220	1410.66	0.0002048	0.0	0.0
Presolved break based	11436	8095	11811	576.88	0.0098	0.0	0.0
la05_0_r_m	dual =	110000.0	opt =	111300.0	gap	= 0.012	
State-Based	6792	9191	11636	3600.03	0.04234	0.0	0.02648
Break based	13825	7076	3730	3600.08	0.01203	0.0008622	0.008844
SCIP+	6641	600	1804	547.13	0.01071	0.0	0.0
SCIP+: col. gen.	6105	645	1344	465.79	0.00128	0.0	0.0
Presolved break based	7756	7110	92371	1543.65	0.012	0.0	0.0
la05_0_r_s	dual =	192100.0	opt =	193400.0	gap	= 0.0067	
State-Based	5087	7987	65150	3600.06	0.01749	0.0	0.0005018
Break based	7084	5394	34117	2018.26	0.006487	0.0	0.0
SCIP+	3239	519	1385	261.51	0.005981	0.0	0.0
SCIP+: col. gen.	3671	588	251	38.65	0.0001023	0.0	0.0
Presolved break based	3523	5415	14439	106.62	0.0063	0.0	0.0
la05_0_s_h	dual =	57140.0	opt =	57810.0	gap	= 0.012	
State-Based	10423	11207	29652	3600.21	0.02574	0.006261	0.02001
Break based	32087	10583	115	3601.09	0.01163	0.01401	0.02448
SCIP+	28811	765	1297	1682.91	0.01096	0.0	0.0
SCIP+: col. gen.	16358	770	7131	3600.02	0.0007318	0.000467	0.009493
Presolved break based	31294	10620	79235	2443.09	0.012	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la05_0_s_l	dual =	62320.0	opt =	63120.0	gap	= 0.013	
State-Based	8883	9597	29653	3603.58	0.03599	0.007922	0.03044
Break based	23917	9140	1012	3600.71	0.01263	0.01332	0.02431
SCIP+	21089	695	2603	3600.1	0.01176	0.0	0.003105
SCIP+: col. gen.	14308	700	7464	3600.02	0.0002489	0.001632	0.004986
Presolved break based	23317	9177	88128	3411.19	0.013	0.0	0.0
la05_0_s_m	dual =	67350.0	opt =	68150.0	gap	= 0.012	
State-Based	7233	7872	18568	3600.03	0.03821	0.001511	0.02361
Break based	16222	7565	1076	3600.36	0.0118	0.01398	0.02449
SCIP+	13738	620	808	1357.22	0.01019	0.0	0.0
SCIP+: col. gen.	9332	625	3364	1910.67	0.0002413	0.0	0.0
Presolved break based	15665	7602	58814	1796.0	0.012	0.0	0.0
la05_0_s_s	dual =	68100.0	opt =	68550.0	gap	= 0.0066	
State-Based	5707	6276	29840	3600.2	0.02108	0.003282	0.01419
Break based	10238	6106	16495	3603.53	0.006628	0.00229	0.006917
SCIP+	7894	550	1415	820.29	0.00615	0.0	0.0
SCIP+: col. gen.	5629	555	1060	305.49	0.0001244	0.0	0.0
Presolved break based	9422	6133	76837	847.82	0.0066	0.0	0.0
la05_1_l_h	dual =	5101.0	opt =	5101.0	gap	= 0.0	
State-Based	9133	16942	1449	821.69	0.0	0.0	0.0
Break based	20732	9245	1	2589.55	0.0	0.0	0.0
SCIP+	6795	700	26	290.63	0.0	0.0	0.0
SCIP+: col. gen.	6211	835	174	142.68	0.0	0.0	0.0
Presolved break based	7801	9276	1	348.94	0.0	0.0	0.0
la05_1_l_l	dual =	5101.0	opt =	5101.0	gap	= 0.0	
State-Based	7593	14422	3150	491.46	0.0	0.0	0.0
Break based	14355	7775	1	2000.35	0.0	0.0	0.0
SCIP+	4982	630	22	124.77	0.0	0.0	0.0
SCIP+: col. gen.	5369	765	211	207.88	0.0	0.0	0.0
Presolved break based	5915	7805	1	176.09	0.0	0.0	0.0
la05_1_l_m	dual =	5101.0	opt =	5101.0	gap	= 0.0	
State-Based	5954	11733	1901	280.73	0.0	0.0	0.0
Break based	8627	6197	1	1013.55	0.0	0.0	0.0
SCIP+	3732	555	93	100.92	0.0	0.0	0.0
SCIP+: col. gen.	4060	690	147	54.55	0.0	0.0	0.0
Presolved break based	4092	6228	1854	235.17	0.0	0.0	0.0
la05_1_l_s	dual =	∞	opt =	-∞	gap	= 0.0	
State-Based	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SCIP+	3184	665	0.0	0.07	0.0	0.0	0.0
SCIP+: col. gen.	2839	620	1	0.27	0.0	0.0	0.0
Presolved break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la05_1_m_h	dual =	3124.0	opt =	3124.0	gap	= 0.0	
State-Based	9731	14511	37	264.4	0.0	0.0	0.0
Break based	25638	9866	1	3600.21	0.0	0.0144	0.0144
SCIP+	10286	730	36	561.74	0.0	0.0	0.0
SCIP+: col. gen.	7131	805	252	273.31	0.0	0.0	0.0
Presolved break based	12316	9899	1	352.04	0.0	0.0	0.0
la05_1_m_l	dual =	3124.0	opt =	3124.0	gap	= 0.0	
State-Based	8191	12411	1	150.73	0.0	0.0	0.0
Break based	18430	8405	1	3600.23	0.0	0.001921	0.001921
SCIP+	7040	660	28	258.76	0.0	0.0	0.0
SCIP+: col. gen.	6593	735	1560	1084.0	0.0	0.0	0.0
Presolved break based	8524	8441	1	220.22	0.0	0.0	0.0
la05_1_m_m	dual =	3124.0	opt =	3124.0	gap	= -0.0	
State-Based	6543	10163	854	165.02	0.0	0.0	0.0
Break based	11785	6830	1	1632.34	0.0	0.0	0.0
SCIP+	4955	585	20	80.2	0.0	0.0	0.0
SCIP+: col. gen.	4444	660	31	14.49	0.0	0.0	0.0
Presolved break based	5666	6860	1	117.86	0.0	0.0	0.0
la05_1_m_s	dual =	3124.0	opt =	3124.0	gap	= 0.0	
State-Based	5037	8097	1564	125.1	0.0	0.0	0.0
Break based	6626	5351	1	687.22	0.0	0.0	0.0
SCIP+	3210	515	41	56.55	0.0	0.0	0.0
SCIP+: col. gen.	3498	590	23	13.92	0.0	0.0	0.0
Presolved break based	3844	5382	1	55.49	0.0	0.0	0.0
la05_1_r_h	dual =	3381.0	opt =	3381.0	gap	= 0.0	
State-Based	9626	14514	1313	472.58	0.0	0.0	0.0
Break based	25311	9749	1	2135.69	0.0	0.0	0.0
SCIP+	11234	728	59	534.61	0.0	0.0	0.0
SCIP+: col. gen.	6933	807	154	135.32	0.0	0.0	0.0
Presolved break based	13941	9777	1	398.57	0.0	0.0	0.0
la05_1_r_l	dual =	2443.0	opt =	2443.0	gap	= -0.0	
State-Based	8200	11351	1	64.18	0.0	0.0	0.0
Break based	20183	8351	1	2769.69	0.0	0.0	0.0
SCIP+	11369	670	146	364.17	0.0	0.0	0.0
SCIP+: col. gen.	6668	725	332	246.86	0.0	0.0	0.0
Presolved break based	12864	8383	106	647.13	0.0	0.0	0.0
la05_1_r_m	dual =	3230.0	opt =	3230.0	gap	= 0.0	
State-Based	6541	9290	1060	278.6	0.0	0.0	0.0
Break based	13123	6801	172	3398.76	0.0	0.0	0.0
SCIP+	6241	596	62	128.69	0.0	0.0	0.0
SCIP+: col. gen.	5247	651	453	215.68	0.0	0.0	0.0
Presolved break based	6659	6828	1	193.3	0.0	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la05_1_r_s	dual =	3517.0	opt =	3517.0	gap	= -0.0	
State-Based	5234	7022	5518	316.31	0.0	0.0	0.0
Break based	8276	5420	1	520.99	0.0	0.0	0.0
SCIP+	4177	537	25	44.5	0.0	0.0	0.0
SCIP+: col. gen.	3632	572	54	17.98	0.0	0.0	0.0
Presolved break based	4629	5450	106	127.62	0.0	0.0	0.0
la05_1_s_h	dual =	1304.0	opt =	1304.0	gap	= 0.0	
State-Based	10429	11215	141	226.19	0.0	0.0	0.0
Break based	31985	10583	1	3173.98	0.0	0.0	0.0
SCIP+	20786	765	40	742.95	0.0	0.0	0.0
SCIP+: col. gen.	8172	770	172	378.94	0.0	0.0	0.0
Presolved break based	23387	10620	1	531.29	0.0	0.0	0.0
la05_1_s_l	dual =	1304.0	opt =	1304.0	gap	= 0.0	
State-Based	8889	9605	1313	399.34	0.0	0.0	0.0
Break based	23815	9140	1	3158.94	0.0	0.0	0.0
SCIP+	13975	695	484	982.49	0.0	0.0	0.0
SCIP+: col. gen.	6549	700	174	139.5	0.0	0.0	0.0
Presolved break based	16239	9177	1	324.56	0.0	0.0	0.0
la05_1_s_m	dual =	1304.0	opt =	1304.0	gap	= 0.0	
State-Based	7239	7880	362	197.76	0.0	0.0	0.0
Break based	16120	7565	1	775.59	0.0	0.0	0.0
SCIP+	8660	620	24	126.83	0.0	0.0	0.0
SCIP+: col. gen.	5224	625	90	48.16	0.0	0.0	0.0
Presolved break based	10027	7602	1	333.39	0.0	0.0	0.0
la05_1_s_s	dual =	1304.0	opt =	1304.0	gap	= 0.0	
State-Based	5713	6284	1	39.06	0.0	0.0	0.0
Break based	9950	6092	1	175.43	0.0	0.0	0.0
SCIP+	5298	550	23	46.62	0.0	0.0	0.0
SCIP+: col. gen.	4122	555	101	40.83	0.0	0.0	0.0
Presolved break based	6151	6129	1	76.62	0.0	0.0	0.0
la05_7_l_h	dual =	335900.0	opt =	336400.0	gap	= 0.0016	
State-Based	9129	16940	11148	795.94	0.007961	0.0	0.0
Break based	20958	9245	897	674.01	0.001552	0.0	0.0
SCIP+	8925	700	23	37.48	0.0004214	0.0	0.0
SCIP+: col. gen.	6716	835	474	56.98	0.0001651	0.0	0.0
Presolved break based	10860	9282	434	47.93	0.00059	0.0	9.5e-05
la05_7_l_l	dual =	336200.0	opt =	336400.0	gap	= 0.00067	
State-Based	7589	14420	8327	327.45	0.00282	0.0	0.0
Break based	14581	7775	77	300.02	0.0006559	0.0	0.0
SCIP+	6119	630	23	13.66	0.0004038	0.0	0.0
SCIP+: col. gen.	5218	765	198	22.45	0.0001748	0.0	0.0
Presolved break based	7828	7814	1	23.32	0.0	0.0	4.2e-05

instance	vars	cons	nodes	time	relative root	relative primal	gap
la05_7_l_m	dual =	336900.0	opt =	337500.0	gap	= 0.0019	
State-Based	5950	11731	8006	264.45	0.003469	0.0	0.0
Break based	8967	6209	671	181.79	0.001846	0.0	0.0
SCIP+	3970	555	53	17.51	0.001756	0.0	0.0
SCIP+: col. gen.	4193	690	269	24.09	0.0004172	0.0	0.0
Presolved break based	5068	6229	972	26.59	0.0017	0.0	0.0
la05_7_l_s	dual =	∞	opt =	-∞	gap	= 0.0	
State-Based	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SCIP+	3184	665	0.0	0.04	0.0	0.0	0.0
SCIP+: col. gen.	2865	620	1	0.27	0.0	0.0	0.0
Presolved break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
la05_7_m_h	dual =	215300.0	opt =	216000.0	gap	= 0.0034	
State-Based	9729	14510	26630	2011.94	0.008752	0.0	0.0
Break based	25810	9866	27786	3011.39	0.003539	0.0	0.0
SCIP+	15060	730	943	334.3	0.002922	0.0	0.0
SCIP+: col. gen.	8239	805	868	199.1	0.0005258	0.0	0.0
Presolved break based	17436	9899	22372	312.09	0.0035	0.0	0.0
la05_7_m_l	dual =	215400.0	opt =	216000.0	gap	= 0.003	
State-Based	8189	12410	21618	1028.81	0.007063	0.0	0.0
Break based	18602	8405	9414	1436.0	0.002885	0.0	0.0
SCIP+	10296	660	330	103.26	0.002774	0.0	0.0
SCIP+: col. gen.	6656	735	514	127.9	0.0004714	0.0	0.0
Presolved break based	12498	8442	7903	158.69	0.003	0.0	0.0
la05_7_m_m	dual =	215600.0	opt =	216600.0	gap	= 0.0047	
State-Based	6541	10162	83048	3600.1	0.009272	0.0	0.0004011
Break based	12185	6842	56456	3600.95	0.004867	0.001163	0.002932
SCIP+	6374	585	4632	676.94	0.004165	0.0	0.0
SCIP+: col. gen.	5622	660	1004	139.35	0.000317	0.0	0.0
Presolved break based	7964	6867	47958	271.0	0.0048	0.0	0.0
la05_7_m_s	dual =	216300.0	opt =	217700.0	gap	= 0.0066	
State-Based	5035	8096	149233	3600.05	0.008478	0.0	0.001375
Break based	6938	5371	82094	3600.51	0.006683	0.000101	0.002135
SCIP+	3737	514	4952	714.74	0.006041	0.0	0.0
SCIP+: col. gen.	4116	590	6452	538.1	0.0005485	0.0	0.0
Presolved break based	4847	5384	123265	501.09	0.0067	0.0	0.0
la05_7_r_h	dual =	201700.0	opt =	202600.0	gap	= 0.0045	
State-Based	9376	14502	23678	3600.05	0.01449	0.0005231	0.004658
Break based	24867	9444	14945	3600.14	0.004575	0.0	0.00176
SCIP+	13355	726	3145	1214.59	0.004269	0.0	0.0
SCIP+: col. gen.	8922	809	2472	742.55	0.0	0.0	0.0
Presolved break based	15448	9479	72745	762.06	0.0039	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la05_7_r_l	dual =	234000.0	opt =	234700.0	gap	= 0.0029	
State-Based	7902	12992	46294	1472.4	0.005144	0.0	0.0
Break based	17349	8085	3749	1184.37	0.002689	0.0	0.0
SCIP+	9167	650	53	55.02	0.001583	0.0	0.0
SCIP+: col. gen.	5854	745	113	31.89	0.0001542	0.0	0.0
Presolved break based	10890	8119	3397	122.18	0.0021	0.0	0.0
la05_7_r_m	dual =	206600.0	opt =	207600.0	gap	= 0.0049	
State-Based	6599	9349	29312	745.06	0.006409	0.0	0.0
Break based	13189	6892	29591	1598.57	0.004778	0.0	0.0
SCIP+	7699	595	770	164.07	0.003807	0.0	0.0
SCIP+: col. gen.	6126	651	1305	209.58	0.0001789	0.0	0.0
Presolved break based	9155	6923	17379	274.81	0.0039	0.0	0.0
la05_7_r_s	dual =	∞	opt =	-∞	gap	= 0.0	
State-Based	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SCIP+	4705	641	0.0	0.2	0.0	0.0	0.0
SCIP+: col. gen.	3067	596	1	0.34	0.0	0.0	0.0
Presolved break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
la05_7_s_h	dual =	98010.0	opt =	98760.0	gap	= 0.0076	
State-Based	10422	11206	52575	3600.14	0.01121	0.001215	0.003889
Break based	32087	10583	5007	3600.15	0.007596	0.001458	0.008145
SCIP+	23805	765	609	532.25	0.006813	0.0	0.0
SCIP+: col. gen.	12381	770	2434	910.73	0.000314	0.0	0.0
Presolved break based	25739	10620	122100	2867.07	0.0077	0.0	5.1e-05
la05_7_s_l	dual =	98190.0	opt =	98880.0	gap	= 0.007	
State-Based	8881	9591	68140	2805.47	0.008078	0.0	0.0
Break based	23917	9140	28742	3600.18	0.006955	0.0001416	0.003409
SCIP+	17362	695	303	249.89	0.006161	0.0	0.0
SCIP+: col. gen.	9238	700	1104	314.54	0.0002654	0.0	0.0
Presolved break based	18999	9178	74242	905.62	0.0065	0.0	0.0
la05_7_s_m	dual =	98270.0	opt =	98880.0	gap	= 0.0062	
State-Based	7231	7866	30864	767.64	0.007465	0.0	0.0
Break based	16222	7565	38229	3089.61	0.006167	0.0	0.0
SCIP+	11578	620	190	90.45	0.006076	0.0	0.0
SCIP+: col. gen.	7105	625	862	222.77	0.000268	0.0	0.0
Presolved break based	12969	7603	40003	585.72	0.0057	0.0	0.0
la05_7_s_s	dual =	98520.0	opt =	98880.0	gap	= 0.0036	
State-Based	5705	6270	8993	229.82	0.005501	0.0	0.0
Break based	10238	6106	8630	478.14	0.003585	0.0	0.0
SCIP+	7152	550	582	128.73	0.003518	0.0	0.0
SCIP+: col. gen.	5082	555	303	49.99	0.0	0.0	0.0
Presolved break based	8141	6132	8170	64.42	0.0035	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la05_8_l_h	dual =	471200.0	opt =	481700.0	gap	= 0.022	
State-Based	9082	16921	33007	3600.08	0.03959	0.0	0.01712
Break based	20787	9245	14550	3306.11	0.02175	0.0	0.0
SCIP+	6229	700	27	36.99	0.01312	0.0	0.0
SCIP+: col. gen.	6596	835	256	103.85	0.0008765	0.0	0.0
Presolved break based	6721	9278	4576	127.6	0.017	0.0	0.0
la05_8_l_l	dual =	473900.0	opt =	481700.0	gap	= 0.016	
State-Based	7542	14401	46192	3600.09	0.03153	0.0	0.006881
Break based	14307	7775	2238	1417.64	0.0162	0.0	0.0
SCIP+	4992	630	188	50.2	0.01267	0.0	0.0
SCIP+: col. gen.	5504	765	311	100.97	0.001657	0.0	0.0
Presolved break based	5337	7808	5088	109.3	0.011	0.0	0.0
la05_8_l_m	dual =	477500.0	opt =	484300.0	gap	= 0.014	
State-Based	5907	11784	27630	1061.64	0.02876	0.0	0.0
Break based	8747	6209	8824	928.61	0.01409	0.0	0.0
SCIP+	3792	554	496	71.54	0.01349	0.0	0.0
SCIP+: col. gen.	4245	690	1650	178.27	0.002634	0.0	0.0
Presolved break based	3971	6238	3604	115.35	0.016	0.0	0.0
la05_8_l_s	dual =	∞	opt =	$-\infty$	gap	= 0.0	
State-Based	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SCIP+	3184	665	0.0	0.05	0.0	0.0	0.0
SCIP+: col. gen.	2865	620	1	0.26	0.0	0.0	0.0
Presolved break based	0.0	0.0	0.0	0.0	0.0	0.0	0.0
la05_8_m_h	dual =	276400.0	opt =	280200.0	gap	= 0.014	
State-Based	9682	14497	11156	996.59	0.02573	0.0	0.0
Break based	25604	9866	21228	2478.96	0.01354	0.0	0.0
SCIP+	9195	730	368	209.84	0.009721	0.0	0.0
SCIP+: col. gen.	7016	805	155	53.9	0.0007193	0.0	0.0
Presolved break based	10042	9901	10722	140.69	0.013	0.0	0.0
la05_8_m_l	dual =	276500.0	opt =	280200.0	gap	= 0.013	
State-Based	8142	12397	18617	962.54	0.02534	0.0	0.0
Break based	18213	8405	11316	1651.97	0.01339	0.0	0.0
SCIP+	6852	660	185	88.04	0.008848	0.0	0.0
SCIP+: col. gen.	5830	735	60	33.62	0.0009079	0.0	0.0
Presolved break based	7566	8441	7781	80.92	0.013	0.0	0.0
la05_8_m_m	dual =	276600.0	opt =	280200.0	gap	= 0.013	
State-Based	6494	10149	3575	273.54	0.02335	0.0	0.0
Break based	11793	6842	6081	745.8	0.01284	0.0	0.0
SCIP+	4648	585	119	53.06	0.008371	0.0	0.0
SCIP+: col. gen.	4741	660	166	47.43	0.00106	0.0	0.0
Presolved break based	5403	6866	3845	53.42	0.013	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la05_8_m_s	dual =	277000.0	opt =	280200.0	gap	= 0.012	
State-Based	4989	8095	3291	224.98	0.01781	0.0	0.0
Break based	6719	5371	1831	500.69	0.01143	0.0	0.0
SCIP+	3341	514	89	30.19	0.007052	0.0	0.0
SCIP+: col. gen.	3731	590	95	22.03	0.001416	0.0	0.0
Presolved break based	3560	5391	1647	48.24	0.012	0.0	0.0
la05_8_r_h	dual =	159800.0	opt =	162900.0	gap	= 0.019	
State-Based	9620	13790	3579	619.4	0.05965	0.0	0.0
Break based	26483	9772	2901	2052.98	0.01895	0.0	0.0
SCIP+	7471	735	529	113.56	0.008615	0.0	0.0
SCIP+: col. gen.	6281	800	938	107.47	0.0007311	0.0	0.0
Presolved break based	7975	9801	1086	140.97	0.015	0.0	0.0
la05_8_r_l	dual =	314600.0	opt =	322300.0	gap	= 0.024	
State-Based	7747	13287	24732	2030.71	0.05464	0.0	0.0
Break based	16214	7972	2174	1878.36	0.02385	0.0	0.0
SCIP+	6119	645	31	30.21	0.02564	0.0	0.0
SCIP+: col. gen.	6159	750	165	72.8	0.0009848	0.0	0.0
Presolved break based	6834	8009	2106	72.18	0.019	0.0	0.0
la05_8_r_m	dual =	193300.0	opt =	197200.0	gap	= 0.02	
State-Based	6341	9444	57627	858.07	0.04648	0.0	0.0
Break based	12206	6626	22203	1581.43	0.01955	0.0	0.0
SCIP+	6417	590	14012	1616.95	0.03744	0.0	0.0
SCIP+: col. gen.	5153	655	2653	281.8	0.002438	0.0	0.0
Presolved break based	6977	6657	36144	159.2	0.019	0.0	0.0
la05_8_r_s	dual =	330500.0	opt =	333900.0	gap	= 0.01	
State-Based	4769	7385	2445	86.53	0.01308	0.0	0.0
Break based	6908	4880	645	321.7	0.01008	0.0	0.0
SCIP+	3901	518	16	22.95	0.009933	0.0	0.0
SCIP+: col. gen.	3358	587	10	4.22	0.0006582	0.0	0.0
Presolved break based	4206	4897	623	23.8	0.01	0.0	0.0
la05_8_s_h	dual =	112300.0	opt =	112600.0	gap	= 0.0026	
State-Based	10383	11214	591	92.98	0.007428	0.0	0.0
Break based	31702	10583	131	1454.21	0.003033	0.0	0.0
SCIP+	11639	765	51	47.46	0.002296	0.0	0.0
SCIP+: col. gen.	6951	770	27	14.63	0.0009835	0.0	0.0
Presolved break based	11354	10618	14	52.37	0.001	0.0	0.0
la05_8_s_l	dual =	113100.0	opt =	114100.0	gap	= 0.0083	
State-Based	8842	9599	6354	409.91	0.01183	0.0	0.0
Break based	23282	9140	7128	1270.45	0.008344	0.0	0.0
SCIP+	8604	695	208	68.94	0.004872	0.0	0.0
SCIP+: col. gen.	6093	700	397	52.13	0.001378	0.0	0.0
Presolved break based	8586	9176	1173	54.88	0.005	0.0	0.0

instance	vars	cons	nodes	time	relative root	relative primal	gap
la05_8_s_m	dual =	113200.0	opt =	114100.0	gap	= 0.0075	
State-Based	7192	7874	2596	301.63	0.01246	0.0	0.0
Break based	15376	7565	3160	855.16	0.007388	0.0	0.0
SCIP+	6191	620	353	81.92	0.005306	0.0	0.0
SCIP+: col. gen.	5315	625	340	54.49	0.001392	0.0	0.0
Presolved break based	6472	7602	1593	54.9	0.0051	0.0	0.0
la05_8_s_s	dual =	113400.0	opt =	114100.0	gap	= 0.0057	
State-Based	5666	6278	2359	215.25	0.01179	0.0	0.0
Break based	9618	6106	1834	518.31	0.005723	0.0	0.0
SCIP+	4152	550	242	46.17	0.005973	0.0	0.0
SCIP+: col. gen.	4163	555	606	62.18	0.002141	0.0	0.0
Presolved break based	4474	6131	960	39.21	0.0055	0.0	0.0

Bibliography

[AB06] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2006.

[ABG+20] Tobias Achterberg, Robert E. Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32(2):473–506, 2020.

[ABZ88] Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988.

[AC91] David Applegate and William Cook. A computational study of the job-shop scheduling problem. *INFORMS Journal on Computing*, 3:149–156, 05 1991.

[Ach09] Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, TU Berlin, 2009.

[AHS00] J. Marjien Akker, Cor Hurkens, and Martin Savelsbergh. Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing*, 12:111–124, 05 2000.

[AKM05] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[ANCK08] Ali Allahverdi, C.T. Ng, T.C.E. Cheng, and Mikhail Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, 2008.

[ANS00] Alper Atamturk, George Nemhauser, and Martin Savelsbergh. Conflict graphs in solving integer programming problems. *European Journal of Operational Research*, 121:40–55, 02 2000.

[ARPV90] T.S. Abdul-Razaq, C.N. Potts, and L.N. Van Wassenhove. A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Applied Mathematics*, 26(2):235–253, 1990.

[Art13] Christian Artigues. A note on time-indexed formulations for the resource-constrained project scheduling problem. *Operations Research Letters*, 45, 06 2013.

[Art17] Christian Artigues. On the strength of time-indexed formulations for the resource-constrained project scheduling problem. *Operations Research Letters*, 45(2):154–159, 2017.

[Bal75] Egon Balas. Facets of the knapsack polytope. *Mathematical Programming*, 8(1):146–164, 1975.

[Bal79] Egon Balas. Disjunctive programming. In P.L. Hammer, E.L. Johnson, and B.H. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 3–51. Elsevier, 1979.

[Bal85] Egon Balas. *On the facial structure of scheduling polyhedra*, pages 179–218. Springer Berlin Heidelberg, 1985.

[BAPT12] Agostino G. Bruzzone, Davide Anghinolfi, Massimo Paolucci, and Flavio Tonelli. Energy-aware scheduling for improving manufacturing process sustainability: A mathematical model for flexible flow shops. *Cirp Annals-manufacturing Technology*, 61:459–462, 2012.

[BBC+22] Ksenia Bestuzheva, Mathieu Besancon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 8.0. Technical report, Zuse Institute Berlin, 2022.

[BDK13] Andreas Bley, Fabio D'Andreagiovanni, and Daniel Karch. Wdm fiber replacement scheduling. In *Proceedings of INOC 2013*, volume 41, pages 189 – 196, 2013.

[BDP96] Jacek Blazewicz, Wolfgang Domschke, and Erwin Pesch. The job-shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93(1):1–33, 1996.

[Bel57] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[BJN+98] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[BJS94] Peter Brucker, Bernd Jurisch, and Bernd Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49(1):107–127, 1994. Special Volume Viewpoints on Optimization.

[BK97] Ralf Borndörfer and Zoltan Kormos. An algorithm for maximum cliques. *Unpublished working paper, Konrad-Zuse-Zentrum für Informationstechnik Berlin*, 1997.

[BL20] Andreas Bley and Andreas Linß. Job shop scheduling with flexible energy prices and time windows. In *Operations Research Proceedings 2019*, Operations Research Proceedings, pages 207–213. Springer, 06 2020.

[BL23] Andreas Bley and Andreas Linß. Propagation and branching strategies for job shop scheduling minimizing the weighted energy consumption. In *Operations Research Proceedings 2022*, Operations Research Proceedings, pages 573–580. Springer, 08 2023.

[BLPN06] Philippe Baptiste, Philippe Laborie, Claude Le Pape, and Wim Nuijten. Constraint-based scheduling and planning. In *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 761–799. Elsevier, 2006.

[BLSV98] Egon Balas, Giuseppe Lancia, Paolo Serafini, and Alkis Vazacopoulos. Job shop scheduling with deadlines. *Journal of Combinatorial Optimization*, 1:329–353, 12 1998.

[BMAB16] Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(1):1–34, 2016.

[Bow59] Edward H. Bowman. The schedule-sequencing problem. *Operations Research*, 7(5):621–624, 1959.

[Bru02] Peter Brucker. Scheduling and constraint propagation. *Discrete Applied Mathematics*, 123(1):227–256, 2002.

[BS15] Lotte Berghman and Frits C.R. Spieksma. Valid inequalities for a time-indexed formulation. *Operations Research Letters*, 43(3):268–272, 05 2015.

[BŠM+18] Ondřej Benedikt, Přemysl Šůcha, István Módos, Marek Vlk, and Zdeněk Hanzálek. *Energy-Aware Production Scheduling with Power-Saving Modes*, pages 72–81. Springer International Publishing, 2018.

[BSSW06] Ralf Borndörfer, Uwe Schelten, Thomas Schlechte, and Steffen Weider. *A Column Generation Approach to Airline Crew Scheduling*, pages 343–348. 01 2006.

[BSV08] Egon Balas, Neil Simonetti, and Alkis Vazacopoulos. Job shop scheduling with setup times, deadlines and precedence constraints. *J. Scheduling*, 11:253–262, 08 2008.

[BT69] E. Beale and John Tomlin. Special facilities in a general mathematical programming system for nonconvex problems using ordered sets of variables. *Operational Research*, 69:447–454, 01 1969.

[Bun21] Bundesnetzagentur | SMARD.de. Smard - systematische marktanalyse für die elektrizitäts- und gaswirtschaft, 05 2021. 12:35.

[BVS+11] Katharina Bunse, Matthias Vodicka, Paul Schönsleben, Marc Brülhart, and Frank O. Ernst. Integrating energy efficiency performance in production management – gap analysis between industrial needs and scientific literature. *Journal of Cleaner Production*, 19(6):667–679, 2011.

[BZ78] Egon Balas and Eitan Zemel. Facets of the knapsack polytope from minimal covers. *SIAM Journal on Applied Mathematics*, 34(1):119–148, 1978.

[CCZ14] Michele Conforti, Gerard Cornuejols, and Giacomo Zambelli. *Integer Programming*. Springer Publishing Company, Incorporated, 2014.

[CGT96] Runwei Cheng, Mitsuo Gen, and Yasuhiro Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms—i. representation. *Computers & Industrial Engineering*, 30(4):983–997, 1996.

[CGW00] T. C. Edwin Cheng, Jatinder N. D. Gupta, and Guoqing Wang. A review of flow-shop scheduling research with setup time. *Production and Operations Management*, 9(3):262–282, 2000.

[CL95] Yves Caseau and François Laburthe. Improving branch and bound for jobshop scheduling with constraint propagation. In *Combinatorics and Computer Science*, 1995.

[COR+11] Alberto Caprara, Marcus Oswald, Gerhard Reinelt, Robert Schwarz, and Emiliano Traversi. Optimal linear arrangements using betweenness variables. *Mathematical Programming Computation*, 3(3):261, 2011.

[CPW98] Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. *A Review of Machine Scheduling: Complexity, Algorithms and Approximability*, pages 1493–1641. Springer US, Boston, MA, 1998.

[CRd06] Pablo E. Coll, Celso C. Ribeiro, and Cid C. de Souza. Multiprocessor scheduling under precedence constraints: Polyhedral results. *Discrete Applied Mathematics*, 154(5):770–801, 2006. IV ALIO/EURO Workshop on Applied Combinatorial Optimization.

[CS96] Yves Crama and Frits C. R. Spieksma. Scheduling jobs of equal length: complexity, facets and computational results. *Mathematical Programming*, 72(3):207–227, 1996.

[DL05] Jacques Desrosiers and Marco E. Lübbecke. *A Primer in Column Generation*, pages 1–32. Springer US, Boston, MA, 2005.

[DT93] Mauro Dell' Amico and Marco Trubian. Applying tabu search to the jobshop scheduling problem. *Annals of Operations Research*, 41(3):231–252, 1993.

[DTG+13] Min Dai, Dunbing Tang, Adriana Giret, Miguel A. Salido, and W.D. Li. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. *Robotics and Computer-Integrated Manufacturing*, 29(5):418 – 429, 2013.

[DW60] George B. Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 02 1960.

[DW90] Martin. E. Dyer and Laurence. A. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26(2–3):255–270, 02 1990.

[DW19] Xuefeng Ding and Jiang Wu. Study on energy consumption optimization scheduling for internet of things. *IEEE Access*, 7:70574–70583, 2019.

[Etc77] Javier Etcheberry. The set-covering problem: A new implicit enumeration algorithm. *Operations Research*, 25(5):760–772, 1977.

[FP17] Tobias Fischer and Marc Pfetsch. Branch-and-cut for linear programs with overlapping SOS1 constraints. *Mathematical Programming Computation*, 10, 06 2017.

[Fra23] Fraunhofer-Institut für Solare Energiesysteme ISE. https://www.energy-charts.info/charts/, 07 2023. 10:45.

[GDDT16] Christian Gahm, Florian Denz, Martin Dirr, and Axel Tuma. Energy-efficient scheduling in manufacturing companies: A review and research framework. *European Journal of Operational Research*, 248(3):744–757, 2016.

[GHSW20] Kaizhou Gao, Yun Huang, Ali Sadollah, and Ling Wang. A review of energy-efficient scheduling in intelligent production systems. *Complex & Intelligent Systems*, 6(2):237–249, 2020.

[GJ79] Michael R. Garey and David. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, first edition edition, 1979.

[GJR84] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32:1195–1220, 12 1984.

[GJR85] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. Facets of the linear ordering polytope. *Mathematical Programming*, 33:43–60, 09 1985.

[GJS76] Michael R. Garey, David. S. Johnson, and Ravi Sethi. Complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[GKM+15] Gerald Gamrath, Thorsten Koch, Alexander Martin, Matthias Miltenberger, and Dieter Weninger. Progress in presolving for mixed integer programming. *Mathematical Programming Computation*, 7, 06 2015.

[GL10] Gerald Gamrath and Marco E. Lübbecke. Experiments with a Generic Dantzig-Wolfe Decomposition for Integer Programs. In Paola Festa, editor, *Experimental Algorithms*, pages 239–252. Springer Berlin Heidelberg, 2010.

[GNM16] Robert S. Garfinkel, George L. Nemhauser, and Mathematics. *Integer Programming*. Dover Publications, Inc., USA, 2016.

[GNS00] Zonghao Gu, George L. Nemhauser, and Martin W. P. Savelsbergh. Sequence independent lifting in mixed integer programming. *Journal of Combinatorial Optimization*, 4(1):109–129, 2000.

[GO22] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*, 2022.

[GR11] José Gonçalves and Mauricio Resende. A biased random-key genetic algorithm for job-shop scheduling. *AT&T Labs Research Technical Report*, 46, 01 2011.

[Har21] Kai Hardenbicker. Job-Shop-Scheduling mit variablen Energiepreisen und Vorrangbeziehungen. Masterarbeit, Universität Kassel, 2021.

[HDD98] Willy Herroelen, Bert De Reyck, and Erik Demeulemeester. Resource-constrained project scheduling: A survey of recent developments. *Computers and Operations Research*, 25(4):279–302, 1998.

[HG14] Oliver Herr and Asvin Goel. Comparison of two integer programming formulations for a single machine family scheduling problem to minimize total tardiness. *Procedia CIRP*, 19:174–179, 2014.

[HSRH22] Hegen Hegen, Shuangyuan Shi, Danni Ren, and Jinjin Hu. A survey of job shop scheduling problem: The types and models. *Computers & Operations Research*, 142:105731, 2022.

[Jac57] James R. Jackson. Simulation research on job-shop production. *Naval Research Logistics Quarterly*, 4(4):287–295, 12 1957.

[JM99] Anant Singh Jain and Sheik Meeran. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113(2):390 – 434, 1999.

[Joh53] Selmer Martin Johnson. *Optimal Two- and Three-Stage Production Schedules with Setup Time Included*. RAND Corporation, Santa Monica, CA, 1953.

[JSV98] Brigitte Jaumard, Frédéric Semet, and Tsevi Vovor. A generalized linear programming model for nurse scheduling. *European Journal of Operational Research*, 107(1):1–18, 1998.

[KB16] Wen-Yang Ku and J. Christopher Beck. Mixed integer programming models for job shop scheduling: A computational analysis. *Computers & Operations Research*, 73:165–173, 2016.

[KFH22] James Kotary, Ferdinando Fioretto, and Pascal Van Hentenryck. Fast approximations for job shop scheduling: A lagrangian dual deep learning method. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(7):7239–7246, 06 2022.

[KNT98] Diego Klabjan, George Nemhauser, and Craig Tovey. The complexity of cover inequality separation. *Operations Research Letters*, 23:35–40, 08 1998.

[Kou94] Christos Koulamas. The total tardiness problem: Review and extensions. *Operations Research*, 42(6):1025–1041, 1994.

[KOW20] Bernard Knueven, James Ostrowski, and Jean-Paul Watson. On mixed-integer programming formulations for the unit commitment problem. *INFORMS Journal on Computing*, 32(4):857–876, 2020.

[KV12] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms.* Springer Publishing Company, Incorporated, 5th edition, 2012.

[Law84] Stephen Lawrence. Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement). Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, 1984.

[LD60] Alisa H. Land and Alison G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[LD05] Marco E. Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 11 2005.

[LDL+14] Ying Liu, Haibo Dong, Niels Lohse, Sanja Petrovic, and Nabil Gindy. An investigation into minimising total energy consumption and total weighted tardiness in job shops. *Journal of Cleaner Production*, 65:87–96, 2014.

[LK78] Jan Karel Lenstra and Alexander H. G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26:22–35, 1978.

[LLK77] B. J. Lageweg, Jan Karel Lenstra, and Alexander H. G. Rinnooy Kan. Job-shop scheduling by implicit enumeration. *Management Science*, 24(4):441–450, 1977.

[LP97] Young Hoon Lee and Michael Pinedo. Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, 100(3):464–474, 1997.

[LR79] Jan Karel Lenstra and Alexander H.G. Rinnooy Kan. Computational complexity of discrete optimization problems. In *Discrete Optimization I*, volume 4 of *Annals of Discrete Mathematics*, pages 121–140. Elsevier, 1979.

[LW66] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

[LW18] Marco E. Lübbecke and Jonas T. Witt. The strength of dantzig-wolfe reformulations for the stable set and related problems. *Discret. Optim.*, 30:168–187, 2018.

[Man60] Alan S. Manne. On the Job-Shop Scheduling Problem. *Operations Research*, 8(2):219–223, 04 1960.

[Mar01] Alexander Martin. *General Mixed Integer Programming: Computational Issues for Branch-and-Cut Algorithms*, pages 1–25. Springer Berlin Heidelberg, 2001.

[MDG19] Oussama Masmoudi, Xavier Delorme, and Paolo Gianessi. Job-shop scheduling problem with energy consideration. *International Journal of Production Economics*, 216:12–22, 2019.

[MDL23] Mouad Morabit, Guy Desaulniers, and Andrea Lodi. Machine-learning–based arc selection for constrained shortest path problems in column generation. *INFORMS Journal on Optimization*, 5(2):191–210, 2023.

[MJSS16] David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe, and Edward C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

[MR23] Stephen J. Maher and Elina Rönnberg. Integer programming column generation: accelerating branch-and-price using a novel pricing scheme for finding high-quality solutions in set covering, packing, and partitioning problems. *Mathematical Programming Computation*, 2023.

[MSS04] Rolf H. Möhring, Martin Skutella, and Frederik Stork. Scheduling with and/or precedence constraints. *SIAM Journal on Computing*, 33(2):393–415, 2004.

[MSTP15] Gökan May, Bojan Stahl, Marco Taisch, and Vittal Prabhu. Multi-objective genetic algorithm for energy-efficient job shop scheduling. *International Journal of Production Research*, 53(23):7071–7089, 2015.

[NCKL11] Giacomo Nannicini, Gérard Cornuéjols, Miroslav Karamanov, and Leo Liberti. Branching on split disjunctions. 08 2011.

[NM10] Kristian Nolde and Manfred Morari. Electric load tracking scheduling of a steel plant. *Computers & Chemical Engineering*, 34:1899–1903, 11 2010.

[NS05] Eugeniusz Nowicki and Czesław Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8:145–159, 04 2005.

[PH22] Myoung-Ju Park and Andy Ham. Energy-aware flexible job shop scheduling under time-of-use pricing. *International Journal of Production Economics*, 248:108507, 2022.

[Pin08] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall international series in industrial and systems engineering. Springer, 2008.

[PR21] Michal Penn and Tal Raviv. Complexity and algorithms for min cost and max profit scheduling under time-of-use electricity tariffs. *Journal of Scheduling*, 24:1–20, 02 2021.

[PVW85] Chris N. Potts and Luk N. Van Wassenhove. A branch and bound algorithm for the total weighted tardiness problem. *Operations Research*, 33(2):363–377, 1985.

[QS94] Maurice Queyranne and Andreas S. Schulz. Polyhedral approaches to machine scheduling. Technical report, 1994.

[QW91] Maurice Queyranne and Yaoguang Wang. Single-machine scheduling polyhedra with precedence constraints. *Mathematics of Operations Research*, 16(1):1–20, 1991.

[RF81] David Ryan and Brian Foster. An integer programming approach to scheduling. *Computer Scheduling of Public Transport*, 1:269–, 01 1981.

[RM21] Paolo Renna and Sergio Materi. A literature review of energy efficiency and sustainability in manufacturing systems. *Applied Sciences*, 11(16), 2021.

[Sad19] Ruslan Sadykov. *Modern Branch-Cut-and-Price.* Habilitation à diriger des recherches, Université de Bordeaux, 12 2019.

[Sav94] Martin W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *INFORMS Journal on Computing*, 6:445–454, 1994.

[Sch86] Alexander Schrijver. *Theory of Linear and Integer programming.* Wiley-Interscience, 1986.

[SCH+16] Maximilian Selmair, Thorsten Claus, Frank Herrmann, Andreas Bley, and Marco Trost. Job shop scheduling with flexible energy prices. 06 2016.

[Smi56] Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 03 1956.

[SOMGSOM14] Fadi Shrouf, Joaquin Ordieres-Meré, Alvaro García-Sánchez, and Miguel Ortega-Mier. Optimizing the production scheduling of a single machine to minimize total energy consumption costs. *Journal of Cleaner Production*, 67:197–207, 2014.

[SS95] Yuri N. Sotskov and Natalia V. Shakhlevich. NP-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics*, 59(3):237–266, 1995.

[SVDVZ96] Marco Schutten, Steef Van De Velde, and W.H.M. Zijm. Single-machine scheduling with release dates, due dates and family setup times. *Management Science*, 42:1165–1174, 11 1996.

[SW92] Jorge P. Sousa and Laurence A. Wolsey. A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical Programming*, 54(1):353–367, 1992.

[Tal82] F. Brian Talbot. Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management Science*, 28(10):1197–1210, 1982.

[TBBK23] Mark Turner, Timo Berthold, Mathieu Besançon, and Thorsten Koch. Branching via cutting plane selection: Improving hybrid branching, 2023.

[TCH17] Marco Trost, Thorsten Claus, and Frank Herrmann. Master production scheduling and the relevance of included social criteria. *ACC Journal 1803-9782*, 2017:146, 11 2017.

[TCTB13] Andrea Trianni, Enrico Cagno, Patrik Thollander, and Sandra Backlund. Barriers to industrial energy efficiency in foundries: a european comparison. *Journal of Cleaner Production*, 40:161–176, 2013. Special Volume: Sustainable consumption and production for Asia: Sustainability through green design and practice.

[TR15] Rodrigo F. Toso and Mauricio G. C. Resende. A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, 30:81 – 93, 2015.

[TRM+13] Patrik Thollander, Patrik Rohdin, Bahram Moshfegh, Magnus Karlsson, Mats Söderström, and Louise Trygg. Energy in swedish industry 2020 – current status, policy instruments, and policy implications. *Journal of Cleaner Production*, 51:109–117, 2013.

[TV15] Veerle Timmermans and Tjark Vredeveld. *Scheduling with State-Dependent Machine Speed*, pages 196–208. Springer International Publishing, Cham, 2015.

[UM10] Yasin Unlu and Scott J. Mason. Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. *Computers & Industrial Engineering*, 58(4):785–800, 2010.

[VAL94] R.J.M. Vaessens, E.H.L. Aarts, and Jan Karel Lenstra. *Job shop scheduling by local search*. Memorandum COSOR. Technische Universiteit Eindhoven, 1994.

[Van05] François Vanderbeck. *Implementing Mixed Integer Column Generation*, pages 331–358. Springer US, Boston, MA, 2005.

[vdA94] J. Marjien van den Akker. *LP-based solution methods for single machine scheduling problems*. PhD thesis, Eindhoven University of Technology, 1994.

[vdAvHS99] J. Marjien van den Akker, C. P. M. van Hoesel, and M. W. P. Savelsbergh. A polyhedral approach to single-machine scheduling problems. *Mathematical Programming*, 85(3):541–572, 1999.

[Wag59] Harvey M. Wagner. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2):131–140, 06 1959.

[WN14] Laurence.A. Wolsey and George.L. Nemhauser. *Integer and Combinatorial Optimization*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 2014.

[Wol90] Laurence A. Wolsey. Valid inequalities for 0–1 knapsacks and mips with generalised upper bound constraints. *Discrete Applied Mathematics*, 29(2):251–261, 1990.

[Wol97] Laurence A. Wolsey. MIP modelling of changeovers in production planning and scheduling problems. *European Journal of Operational Research*, 99(1):154–165, 1997.

[Wol98] Laurence A. Wolsey. *Integer programming / Laurence A. Wolsey*. Wiley-Interscience series in discrete mathematics and optimization. J. Wiley, New York, 1998 - 1998.

[WS11] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[WX06] Lan Wang and Yang Xiao. A survey of energy-efficient scheduling mechanisms in sensor networks. *MONET*, 11:723–740, 10 2006.

[XSRH22] Hegen Xiong, Shuangyuan Shi, Danni Ren, and Jinjin Hu. A survey of job shop scheduling problem: The types and models. *Computers & Operations Research*, 142:105731, 2022.

[Yan99] Wen-Hwa Yang. Survey of scheduling research involving setup times. *International Journal of Systems Science*, 30(2):143–155, 1999.

[Yin04] Ai-Hua Yin. A heuristic algorithm for the job shop scheduling problem. In Hai Jin, Guang R. Gao, Zhiwei Xu, and Hao Chen, editors, *Network and Parallel Computing*, pages 118–128. Springer, 2004.

[ZDZ+19] Jian Zhang, Guofu Ding, Yisheng Zou, Shengfeng Qin, and Jianlin Fu. Review of job shop scheduling research and its new perspectives under industry 4.0. *Journal of Intelligent Manufacturing*, 30(4):1809–1830, 2019.