Czech technical university in Prague

Faculty of Electrical Engineering
Department of Computer Science

MASTER'S THESIS

Bc. Ondřej Benedikt

Algorithms for Energy-Aware Production Scheduling with Power-Saving Modes

Supervisor: doc. Ing. Přemysl Šůcha, Ph.D.

Study program: Open Informatics

Branch of study: Artificial Intelligence

Prague
2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Benedikt** Jméno: **Ondřej** Osobní číslo: **420076**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Umělá intelligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Algoritmy pro optimalizaci výroby s ohledem na spotřebu s využitím energeticky úsporných režimů strojů

Název diplomové práce anglicky:

Algorithms for Energy-Aware Production Scheduling with Power-Saving Modes

Pokyny pro vypracování:

An efficient way to reduce the energy expenses in production is to turn a machine off when it is not being used or switch it into an energy-saving mode.This can be achieved by an appropriate production schedule that could control the switching between the energy modes with respect to the required production volume. The goal of this thesis is to propose an exact algorithm for this scheduling problem. The particular objectives of the thesis are:
1) Review the existing works in the energy-aware scheduling domain.
2) Design an exact algorithm for the scheduling problem using an appropriate decomposition technique.
3) Implement the algorithm and test it on randomly generated benchmark instances.
4) Compare the proposed algorithm with an alternative state-of-the-art approach.

Seznam doporučené literatury:

[1] Feillet, D.: A tutorial on column generation and branch-and-price for vehicle routing problems. 4OR 8(4) (Dec 2010) 407-424
[2] Shrouf, F., Ordieres-Mere, J., Garcia-Sanchez, A., Ortega-Mier, M.: Optimizing the production scheduling of a single machine to minimize total energy consumption costs. Journal of Cleaner Production 67(Supplement C) (2014) 197-207.
[3] Vaclavik, R., Novak, A., Sucha, P., Hanzalek, Z.: Accelerating the branch-and-price algorithm using machine learning. European Journal of Operational Research (Oct 2017) under review.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Přemysl Šůcha, Ph.D., optimalizace CIIRC

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **19.02.2018** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **30.09.2019**

doc. Ing. Přemysl Šůcha, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

Datum převzetí zadání

Podpis studenta

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne ..... Podpis autora

Anotace

Efektivní rozvrhování výroby může mít výrazný vliv na celkové náklady podniku. Jelikož celosvětová spotřeba energie stále roste, je také potřeba zabývat se problémy spojenými s optimalizací energetické spotřeby, abychom dosáhli dlouhodobě udržitelné produkce. Cílem této práce je navrhnout přístupy řešení pro optimalizaci provozních nákladů méně využitých strojů pomocí změny jejich provozních stavů. Konkrétně se zde jedná o problém paralelních identických strojů, na které jsou rozvrhovány jednotlivé úlohy. Jsou navrženy dva přístupy pro nalezení optimálního řešení problému. První přístup modeluje problém jako celek (vzniká tak tzv. globální model), zatímco druhý problém dekomponuje pomocí Dantzig-Wolfeho techniky. Cílem dekompozice je odstranit symetrie, které se objevují v globálním modelu kvůli zaměnitelnosti jednotlivých strojů. Jsou použity dva možné způsoby formulace modelů, a to MILP a CP. Navíc je implementován referenční MILP model, který byl vybrán z relevantních zdrojů. Je provedena řada experimentů, které detailně porovnávají navržené přístupy, a jejich výsledky jsou podrobně popsány.

Klíčová slova

Celočíselné lineární programování, Programování s omezujícími podmínkami, Dantzig-Wolfeho dekompozice, Branch-and-price, Optimalizace energetické spotřeby, Rozvrhování

Annotation

An efficient production scheduling can have a significant impact on the total production costs. Moreover, global energy consumption has been steadily rising, and so it is important to tackle the problem of energy optimization to achieve a sustainable production in the long term. The aim of this work is to optimize operation costs (energy consumption) of under-utilized machines by changing their operation modes. Specifically, the problem with parallel identical machines and jobs characterized by their release times, deadlines and processing times is addressed. Two exact approaches are developed to solve the problem. At first, a single global model is formulated to describe the problem as a whole. Afterwards, it is decomposed by Dantzig-Wolfe technique to eliminate symmetries arising due to the interchangeability of the parallel machines. Two formulations frameworks, namely MILP and CP, are tested. Besides, a reference MILP model is adapted from the literature for comparison. Several experiments are conducted, and the results are discussed in detail.

Keywords

Mixed Integer Linear Programming, Constraint Programming, Dantzig-Wolfe Decomposition, Branch-and-Price, Energy Optimization, Scheduling

Acknowledgements

Firstly, I would like to express my gratitude to my supervisor doc. Ing. Přemysl Šůcha, Ph.D. for his support and various insightful discussions.

My sincere thanks also go to Ing. István Módos for his valuable and constructive suggestions, which helped to make this work better.

I am also particularly grateful for stylistic consultations and expert knowledge of CP optimizer provided by Mgr. Marek Vlk.

Last but not least, I would also like to thank my family for their support and encouragement throughout my study.

Contents

1 Prologue

1.1 Introduction

1.1.1 Field of study

The global energy consumption has been steadily rising, as shown in Figure 1.1. Between 1965 and 2015, the amount of consumed energy more than tripled. According to the Energy Regulatory Office, more than 30 % of energy consumed in the Czech Republic were spent on the industrial sector [1]. In this particular sector, energy is used for various purposes, such as for material processing and product assembly, heating and cooling, lighting, operating industrial motors and machinery, air conditioning etc. To achieve a sustainable development, people need to find ways how to save energy. One of the measures, which could be taken, is the efficient scheduling of energy demanding machines.

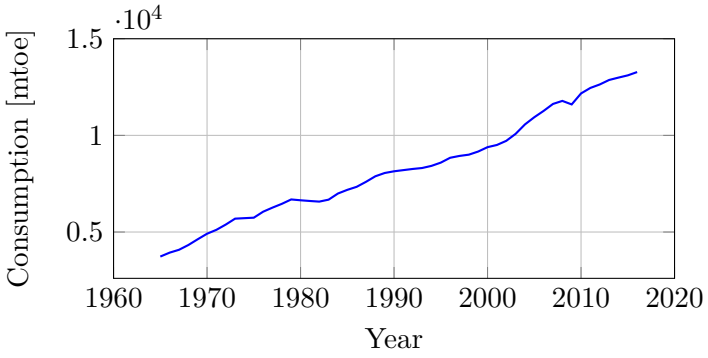


Figure 1.1: The global energy consumption between 1965 and 2016 in millions of tonnes of oil equivalent (mtoe) [2]

From a different point of view, automated production scheduling is becoming an integral part of the industry today, bringing many benefits, such as improving the efficiency of production either regarding the speed or the cost. The largest economy in Europe, Germany, initiated movement called 'Industry 4.0', which is commonly referred to as 'the fourth industrial revolution'. It addresses problems of connectivity, cloud computing, decentralized decision making and information transparency and also problems of automation and data exchange in manufacturing technologies. From this perspective, the automated scheduling for energy optimization is not only the interesting research topic but also an important part of the industry today and in the future.

However, scheduling problems are often hard to solve and so specialized algorithmic ap-

proaches need to be developed. This work concentrates mainly on the production process, where machines with a high energy consumption are used. Several sources, such as [3, 4, 5], mention that substantial energy cost savings, up to tens of percent, can be achieved by managing the state of the machine in time. For example, a machine can be turned off (or to some power-saving mode) when nothing is being processed. However, the transitions between the states might take some time and even consume energy. Therefore the switching between modes has to be planned carefully. The objective is to find a schedule of the jobs and a switching between power modes of the identical, parallel machines so that the total energy consumption is minimized.

The inspiration for this work came from two real production processes with high energy demands. One of them is a glass tempering in ERTL Glas company, while the other one is a steel hardening in ŠKODA AUTO company. Both processes have in common that material is heated in one of several identical furnaces. The temperature inside has to be very high, up to hundreds of degrees Celsius – depending on the particular technological process. In consequence, energy consumption is also high. Typically, all the furnaces continuously operate from the beginning to the end of the scheduling horizon (as this strategy is easy to deploy and does not need specialized workers/software, who/which would find the energy-saving schedule). In consequence, the high energy-demanding state is not changed, even if nothing is being produced, thus wasting energy. The preliminary feasibility study for ŠKODA AUTO Company [6] has shown that about 6 % of the production line consumption could be saved using the power-saving modes.

1.1.2 Related work

There has been a growing interest in the energy-efficient scheduling in the last few decades. Authors are interested in the topic mainly for two reasons; the first one is environmental, whereas the second one is economical. As reported by the International Energy Agency, the manufacturing industry sector is responsible for about one-third of primary energy consumption. Furthermore, the CO_2 emissions from industry accounted for 36 % of total global CO_2 emissions in 2017 [7]. As for the economic aspect, energy-use during the production can be improved by two measures: technological and organizational [8]. Technological improvements are usually associated with large-scale investments, because of the high cost of development and machines. On the other hand, organizational measures can improve energy efficiency at a low cost; one example of such a measure would be the poduction scheduling.

In 2016, an extensive review of energy-efficient scheduling in manufacturing companies was published [3]. In total, authors categorized 87 relevant articles, which were published between 1990 and 2015. The study shows that (i) the number of published papers on the energy-efficient scheduling and sustainable manufacturing has been increasing and (ii) the energy savings up to tens of percents can be achieved. The primary focus of the researchers is on the flow-shop problems, however, 15 surveyed papers studied the problem

with parallel machines.

One of the first works analysing under-utilized resources and considering machine modes to minimize a total energy consumption was [4]. Authors provided a detailed performance analysis indicating that large quantities of energy are consumed by non-bottleneck machines during their idle times. Authors also proposed a multi-objective Integer Linear Programming model optimizing weighted total completion time and energy consumption. However, it was assumed, that jobs are processed in the order of their arrival, which makes the model significantly easier to solve.

Mitra et al. [5] have been studying the minimization of operating expenses under time-dependent electricity pricing (TOU) for continuous production planning. Their research was inspired by real-world scheduling for cement plants. Problem studied in this thesis is a bit different as time-dependent prices and continuous production are not taken into account. Instead, individual jobs with processing times depending on a machine-mode are scheduled here.

Shrouf et al. [9] proposed a mathematical model (and a heuristic algorithm) to minimize energy consumption for a single machine production scheduling. Again, variable prices were assumed. Contrary to [4], no assumptions about the order of the jobs were given. Their work was directly extended by [10], who modelled a job shop production system (parallel machines) minimizing energy with flexible energy prices.

The optimal solutions of both to the previously mentioned works are based on time-indexed Linear Integer Programming models, which can optimally solve only small instances in a reasonable time. Furthermore, authors assume only several modes, such as *off*, *standby*, *on*, *ramp-up* and *ramp-down*. Contrary to that, one of the aims of this thesis is to formulate an efficient and universal model, meaning that multiple modes and transitions between them can be defined and that the model will work reasonably well even for larger instances.

Gong et al. [11] extensively studied a single-resource scheduling problem. They use a finite state automaton (FSM) to describe a state of a resource. However, authors propose only three types of machine operations, namely *immediately start next job*, *stay idle* and *turn off*.

The problem studied here, i.e., the parallel-machine scheduling with mode-dependent processing times, is similar to a dynamic voltage scaling in embedded systems [12]. In their setting, there are multiple frequencies of the processor and the jobs (tasks to be processed) have different processing time for each such a frequency. But since the schedules in the embedded systems are usually event-triggered and transition times between two operating frequencies are negligible, the research cannot be directly applied to the production process.

For clarity, selected authors working on the energy optimization scheduling problems are also listed in Table 1.1. Many more works have been published on energy optimization for parallel-machine scheduling; however, to the best of my knowledge, none of them

explicitly models machine-modes and transitions between them using more modes than just *processing* and *idle*.

1.1.3 Contribution

A problem of scheduling jobs on multi-mode, parallel, identical machines while optimizing energy consumption is described in detail. Furthermore, exact algorithms for the problem-solving are proposed in this work. Specifically, two approaches are described; the first one uses the global model, solving the problem as it is, while the second one is based on a decomposition technique, designed to get rid of the symmetries arising from the interchangeability of the machines. Moreover, two model formulations are compared – one based on the Integer Linear Programming (MILP) and the other one based on the Constraint Programming (CP). The algorithms are tested on randomly-generated benchmark instances and compared with a reference model adopted from the literature.

1.2 Outline

The main text is divided into three parts. The problem is formally defined in a chapter named Theoretical background. In that chapter, techniques of MILP solving are briefly introduced too, as well as the principles of Dantzig-Wolfe decomposition. The second part, named Models, contains descriptions of all used models, specifically the global models (MILP and CP), the decomposed models and a reference model. Finally, three experiments are described in the last chapter, and their results are discussed in detail.

Author(s)	Year	Modes	Parallel	Note
Boukas et al. [13]	1991	✗	✓	a non-optimal approach minimizing due date of the energy-constrained production schedule
Mouzon et al. [4]	2007	✓	✗	analysis and one of the first systematic attempts to save energy by scheduling under-utilized resources
Mitra et al. [5]	2012	✓	✓	an industrial case study and an optimal solution (MILP) to a continuous-production problem
Moon et al. [14]	2013	✗	✓	a genetic algorithm for the non-optimal solution of a multi-objective optimization problem with 3 rates of energy prices
Fang et Lin [12]	2013	✓	✓	an optimal MILP model and a non-optimal particle-swarm optimization algorithm for weighted tardiness/cost problem; dynamic voltage scaling for embedded systems with negligible transition times
Artigues et al. [15]	2013	✗	✓	a hybrid non-optimal MILP/CP algorithm; processing times of jobs depend on the power input
Shrouf et al. [9]	2014	✓	✗	an optimal MILP model for a single machine problem with TOU pricing
Selmair et al. [10]	2016	✓	✓	a time-indexed MILP model for a job-shop problem with several states of the resources
Gong et al. [11]	2016	✓	✗	a FSM is used for modes modelling, but possible transitions between modes are limited to only several options
Che et al. [16]	2017	✗	✓	a MILP model and a two-stage heuristic for an unrelated-parallel-machine problem with TOU pricing scheme

Table 1.1: Selected authors working on the energy optimization scheduling listed in chronological order; column *Modes* indicates whether machine modes were modelled, column *Parallel* indicates whether parallel resources were used

2 Theoretical background

2.1 Problem statement

Being inspired by the two production processes, the problem statement is formulated as follows: scheduling jobs on identical, parallel machines, where each job is characterized by its release time, deadline and processing time(s). In some production processes, machines operating in a power-saving mode can still process material at the cost of longer processing time. In order to take this into consideration, it is assumed that the processing time of a job is dependent on the mode in which a machine is operating. The objective is to find a schedule of the jobs and a schedule of the machines, i.e., switching between their modes, such that the total energy consumption (cost) is minimized.

This high-level description is formalized in the following paragraphs. Note, that domains of several parameters, such as processing times, will be integral, not real-valued. That is because real production scheduling problems are usually discretized (by minutes, 15-minutes intervals, hours or even days – depending on an application area).

2.1.1 Resources

Let $\mathcal{M} = \{1, 2, \ldots, M\}$ be a set of parallel, identical machines. Each of these machines operates on a finite, directed *transition graph* $G = (\mathcal{V},\ \mathcal{E})$, where $\mathcal{V} = \{1, 2, \ldots, V\}$ is a set of its vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of its edges. The vertices correspond to the possible modes of a machine, while the edges correspond to the transitions between these modes. If there is no edge between two vertices v, v', it means that the immediate transition between modes v and v' is not possible. There are two special vertices $v_{\text{init}} \in \mathcal{V}$ and $v_{\text{term}} \in \mathcal{V}$ representing the *initial mode* and the *terminal mode* of the machines, respectively.

As a transition from mode v to mode v' may take some time, *transition time* is defined as $\mathrm{t}_{v,v'} \in \mathbb{N}_0 \cup \{\infty\}$, where $\mathrm{t}_{v,v'} = \infty$ means, that the transition between v and v' is not possible (no edge in the transition graph). A machine is not operational during the transition, yet it may still consume energy. Therefore *transition cost* $\mathrm{c}_{v,v'} \in \mathbb{R}_0^+ \cup \{\infty\}$ is defined. A machine starts operating in mode v' immediately after the transition from v to v' is completed.

While a machine is operating in mode $v \in \mathcal{V}$, it demands constant *power* $\mathrm{w}_v \in \mathbb{R}_0^+$. If it is operating in some mode v for total time t, *operating cost* is computed as $\mathrm{w}_v \cdot t$.

Sometimes, it is not desirable to spend too much time in a mode, usually because of the used technology (the machine could overheat or have some other technical issues). Similarly, spending too little time in a mode could be technologically infeasible or simply unprofitable. So $\mathrm{t}_v^{\min} \in \mathbb{N}_0$ and $\mathrm{t}_v^{\max} \in \mathbb{N}_0$ are defined to model a *minimal* and a *maximal time* for which mode v can be operating continuously. It means that at most after $\mathrm{t}_v^{\max}$ time units are spent in mode v, a transition to a different mode has to occur (or scheduling horizon has to be reached) and similarly before $\mathrm{t}_v^{\min}$ time units are spent in mode v, no transition can occur. The maximal number of transitions that can happen on a single machine during a scheduling horizon is limited by constant $(i_{\max} - 1) \in \mathbb{N}$, where $i_{\max}$ corresponds to the maximal length of a single scheduling profile, see Section 2.1.3; denoting $\mathcal{I} = \{1, \ldots, i_{\max}\}$. This limitation is introduced because of the technological reasons, as an extensive switching between the individual modes could damage the machine.

2.1.2 Jobs

Let $\mathcal{J} = \{1, 2, \ldots, J\}$ be a set of jobs. Each job has to be processed on some machine within the scheduling horizon $\mathrm{h} \in \mathbb{N}$. Once the processing starts, it cannot be preempted. Each machine can process at most one job at a time and it cannot change its working mode while processing. Jobs cannot be processed on a machine during the time of its transition from one mode to another.

A *processing time* of job $j \in \mathcal{J}$ depends on the mode $v \in \mathcal{V}$, in which the assigned machine operates while processing the job. It is denoted as $\mathrm{p}_{j,v} \in \mathbb{N}_0 \cup \{\infty\}$. If $\mathrm{p}_{j,v} = \infty$ for some job j and mode v, it means that job j cannot be processed while the assigned machine is operating in mode v. Each job $j \in \mathcal{J}$ also has a *release time* and a *deadline*, denoted as $\mathrm{r}_j \in \mathbb{N}_0$ and $\mathrm{d}_j \in \mathbb{N}_0$, respectively. These form a time window within which the job has to be processed.

2.1.3 Solution

A *solution* consists of a schedule of the jobs and a schedule of the machine modes. Formally, it is a tuple $(a, s, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_M, \boldsymbol{t}_1^{\mathrm{mode}}, \ldots, \boldsymbol{t}_M^{\mathrm{mode}})$, where $a : \mathcal{J} \to \mathcal{M}$ is a function representing the *job assignment to the machines*, $s : \mathcal{J} \to \{0, 1, \ldots, \mathrm{h}\}$ is a function mapping jobs onto their *start times*, $\boldsymbol{\pi}_m \in \bigcup_{l \in \{1,2,\ldots,i_{\max}\}} \mathcal{V}^l$ is a *profile* of machine $m \in \mathcal{M}$ and $\boldsymbol{t}_m^{\mathrm{mode}} \in \mathbb{N}_0^{|\boldsymbol{\pi}_m|}$ are *the operating times* of machine m.

Profile $\boldsymbol{\pi}_m$ is a finite sequence of the modes which is followed by machine m in the solution; its length is denoted by $|\boldsymbol{\pi}_m|$. It represents only the transitions between modes; *operating times* of the modes on machine m are captured by $\boldsymbol{t}_m^{\mathrm{mode}}$. Symbol $\pi_{m,i}$ is used to address i-th mode on m-th machine. The time spent in this mode is denoted as $t_{m,i}^{\mathrm{mode}}$.

A feasible solution is a solution, which respects restrictions defined above in sections 2.1.1 and 2.1.2 (such as that jobs do not overlap on a single resource, a machine can not

change its mode while processing some job, etc.). Furthermore, schedules of the individual machines should cover the whole scheduling horizon – the first mode has to start at time 0 while the last one has to end at time h, and the sum of all operating times plus the sum of all transition times must be equal to h for each resource m, i.e.

$$\sum_{i=1}^{|\boldsymbol{\pi}_m|-1} \mathrm{t}_{\pi_{m,i},\pi_{m,i+1}} + \sum_{i=1}^{|\boldsymbol{\pi}_m|} t^{\mathrm{mode}}_{m,i} = \mathrm{h}\,, \forall\, m \in \mathcal{M}. \tag{2.1}$$

In addition, all the profiles have to start with the initial mode v_{init} and end with the terminal mode v_{term}. Thanks to this, one can have control over the initial and terminal state of the machines.

2.1.4 Objective

A goal of this scheduling problem is to find a schedule minimizing the *total cost* (energy consumption), i.e., the sum of transition costs and operating costs computed over all machines

$$\sum_{m\in\mathcal{M}} \left(\sum_{i=1}^{|\boldsymbol{\pi}_m|-1} \mathrm{c}_{\pi_{m,i},\pi_{m,i+1}} + \sum_{i=1}^{|\boldsymbol{\pi}_m|} \mathrm{w}_{\pi_{m,i}} \cdot t^{\mathrm{mode}}_{m,i} \right). \tag{2.2}$$

An optimal solution to this scheduling problem is such a feasible solution that minimizes the total cost (2.2).

2.1.5 Example

The problem statement above may seem heavy in notation. However, all of the constraints and requirements are natural for the production scheduling problems. An example is described here to illustrate the problem and its solution. Note, that all parameters are simple, selected specially for illustration purposes.

Let us assume there are 2 parallel, identical machines, $\mathcal{M} = \{1, 2\}$. Their operating modes are depicted in Figure 2.1. Let us have 4 jobs to be scheduled, $\mathcal{J} = \{1, 2, 3, 4\}$. Their release times, deadlines and processing times are given by Table 2.1.

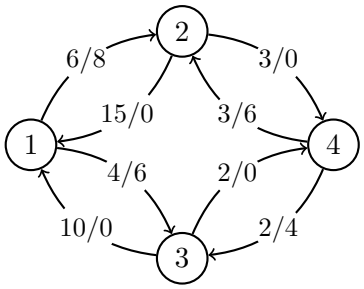
The initial and terminal mode is mode 1, which has zero power consumption, representing a state, in which a machine is turned off. Mode 4 represents *standby* mode. If a machine is operating in this mode, no job can be processed, but the machine is not turned off, so it will be faster to resume the production while saving some energy by lowering the consumed power. Modes 2 and 3 are processing modes, i.e., modes in which the jobs will be processed. Note that because there are no edges between mode 1 and 4 in the transition graph, immediate transitions between these two modes are prohibited.

v	$\mathrm{t}_v^{\min}$	$\mathrm{t}_v^{\max}$	w_v
1	0	∞	0
2	0	10	10
3	0	∞	8
4	5	∞	5

(a)

	1	2	3	4
1	∞	6/8	4/6	∞
2	15/0	∞	∞	3/0
3	10/0	∞	∞	2/0
4	∞	3/6	2/4	∞

(b)



(c)

Figure 2.1: Description of the example transition graph: (a) modes $v \in \mathcal{V}$ and their parameters, (b) transition time / transition cost and (c) graph visualization

j	r_j	d_j	$p_{j,1}$	$p_{j,2}$	$p_{j,3}$	$p_{j,4}$
1	6	20	∞	5	∞	∞
2	8	30	∞	7	∞	∞
3	18	35	∞	∞	10	∞
4	30	50	∞	∞	12	∞

Table 2.1: Jobs $j \in \mathcal{J}$ and their parameters

The length of the scheduling horizon is $\mathrm{h} = 65$, and the schedule must have at most four transitions, $i_{\max} = 5$. One example of a feasible solution is shown in Figure 2.2. The total cost of this solution is

$$\underbrace{(5 \cdot \mathrm{w}_2 + 39 \cdot \mathrm{w}_1)}_{oper_1} + \underbrace{(\mathrm{c}_{1,2} + \mathrm{c}_{2,1})}_{trans_1} + \underbrace{(2 \cdot \mathrm{w}_1 + 7 \cdot \mathrm{w}_2 + 5 \cdot \mathrm{w}_4 + 22 \cdot \mathrm{w}_3 + 8 \cdot \mathrm{w}_1)}_{oper_2} + \underbrace{(\mathrm{c}_{1,2} + \mathrm{c}_{2,4} + \mathrm{c}_{4,3} + \mathrm{c}_{3,1})}_{trans_2},$$

which is exactly the sum of the total operating cost $oper_1$ and transition cost $trans_1$ of the first machine plus the operating cost $oper_2$ and transition cost $trans_2$ of the second machine. Note that this feasible solution is optimal. Jobs 1 and 2 cannot be processed immediately one after another, because there is a limit $\mathrm{t}_2^{\max} = 10$.

The solution depicted by Figure 2.2 is characterized by tuple $(a, s, \boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \boldsymbol{t}_1^{\mathrm{mode}}, \boldsymbol{t}_2^{\mathrm{mode}})$,

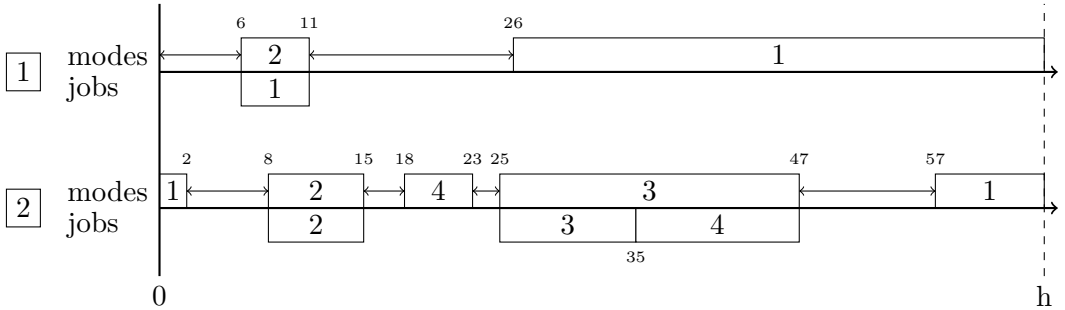


Figure 2.2: Example solution – jobs and modes assignment to the resources

j	$a(j)$	$s(j)$
1	1	6
2	2	8
3	2	25
4	2	35

Table 2.2: Assignments $a(j)$ and start times $s(j)$ of individual jobs j

where functions a, s are described by Table 2.2 and $\boldsymbol{\pi}_1 = (1, 2, 1)$, $\boldsymbol{\pi}_2 = (1, 2, 4, 3, 1)$, $\boldsymbol{t}_1^{\text{mode}} = (0, 5, 39)$, $\boldsymbol{t}_2^{\text{mode}} = (2, 7, 5, 22, 8)$.

2.2 Complexity

It is easy to show, that the standard problem $1 \mid r_j, \tilde{d}_j \mid C_{\max}$ (minimization of the schedule length for a problem with monoprocessor and jobs characterized by release times, deadlines and processing times), can be polynomially transformed to the problem defined here. The trick is to use a transition graph shown in Figure 2.3 and to set $v_{\text{init}} = \text{proc}$, $v_{\text{term}} = \text{off}$, $i_{\max} = 2$, $\text{h} = \max_j \tilde{d}_j$. Furthermore, jobs are allowed to be processed only in v_{proc}; the original release times, deadlines and processing times remain the same. The total cost is minimized, which can be written as (2.3).

$$\underset{t_{\text{proc}},\, t_{\text{off}}}{\text{minimize}}\ t_{\text{proc}} \cdot \underbrace{\text{w}_{\text{proc}}}_{1} + \underbrace{\text{t}_{\text{proc,off}}}_{0} + \underbrace{t_{\text{off}} \cdot \text{w}_{\text{off}}}_{0} = \text{minimize}\ t_{\text{proc}} \tag{2.3}$$

So the minimization of the total cost reduces to a minimization of the time spent in the processing mode. The part of an optimal schedule corresponding to the processing mode directly represents the solution to $1 \mid r_j, \tilde{d}_j \mid C_{\max}$. The processing mode has to start at time 0, and so minimization of t_{proc} corresponds to minimization of $C_{\max}$.

This was for the optimization problems. Of course, it would be very similar for their decision variants. It was proven, that $1 \mid r_j, \tilde{d}_j \mid C_{\max}$ is strongly $\mathcal{NP}$-hard (by reduction from 3-partition problem, which is $\mathcal{NP}$-complete [17]), and so the problem studied here is also $\mathcal{NP}$-hard.

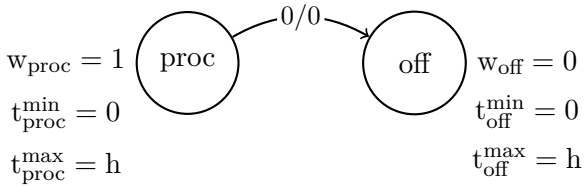


Figure 2.3: Transition graph used for the polynomial reduction

2.3 Brief introduction to LP/MILP solution approaches

In this section, basic terminology used in the field of mathematical (specifically linear) programming is defined. Some classical solution approaches are also briefly described together with a decomposition principle, which is often used to improve (make faster) the solving process of structured linear programs.

Inspiration came from several sources: Gerald Gamrath describes a branch-and-price algorithm in detail in his diploma thesis [18], the algorithm and its application are also described in the tutorial [19]. Some parts of this section are inspired by [20] (mainly the overview of branch-and-price) and by [21] (branch-and-bound procedure and cutting planes technique). Details about Dantzig-Wolfe decomposition and its connection to the column generation approaches can be found in [22, 23], among others.

Note that notation used in this section is unrelated to the notation established in Section 2.1. For example, symbol j can be used as a generic index; it does not represent a job. It is because a general notation in this section is not directly related to the problem and it would become over-complicated if new symbols were used. Also note, that expression $\boldsymbol{x} \geq \boldsymbol{0}$ is used with the interpretation *all elements of vector* $\boldsymbol{x}$ *are non-negative.*

2.3.1 Overview

Linear Program

A *linear program* (LP) is an optimization problem that minimizes (or maximizes) a linear objective function constrained by a set of linear inequalities. Formally, a linear program can be written in the form

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^T\boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}, \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{aligned} \tag{2.4}$$

where $\boldsymbol{A} \in \mathbb{R}^{(m \times n)},\ \boldsymbol{b} \in \mathbb{R}^m,\ \boldsymbol{c} \in \mathbb{R}^n,\ \boldsymbol{x} \in \mathbb{R}^n,\ n \in \mathbb{N},\ m \in \mathbb{N}$. A set of solutions to a linear program (2.4) is called *a polyhedron*, i.e., $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}\}$.

Note that multiple similar formulations of the LP problem can be seen in the literature; for example maximization problems, problems with equalities or problems with unbounded vector $\boldsymbol{x}$. However, all of these formulations are equivalent to (2.4) in the sense that one can be easily translated to the other one while preserving the objective value of the optimal solution. For example, $\max \boldsymbol{c}^T\boldsymbol{x}$ can be transformed to $\min -\boldsymbol{c}^T\boldsymbol{x}$, equality $\boldsymbol{a}\boldsymbol{x} = b$ ($\boldsymbol{a}$ being a row vector of matrix $\boldsymbol{A}$) can be split into two inequalities $\boldsymbol{a}\boldsymbol{x} \leq b$ and $\boldsymbol{a}\boldsymbol{x} \geq b$ (which is the same as $-\boldsymbol{a}\boldsymbol{x} \leq -b$), etc. For further details see, e.g., [24].

There exist multiple methods for solving the linear programming problems. The *ellipsoid method* [25] and the *interior point method* [26] are two examples of methods with a polynomial time-complexity. The *simplex method* [27], which has an exponential time-complexity in general, is also often used because it works very well for the practical problems.

Mixed Integer Linear Program

The formulation of a *mixed integer linear program* (MILP) is very similar to (2.4). Practically, the only difference to LP is that for MILP there exists a non-empty set of indices $I \subseteq \{1, \ldots, n\}$, where n corresponds to a number of variables in the MILP problem, such that variables associated with those indices are integral, i.e., $x_i \in \mathbb{Z},\ \forall i \in I$.

Whereas the formulation is very similar, the complexity is not. Adding integrality constraints makes the problem significantly harder to solve; MILP is $\mathcal{NP}$-hard [28].

Two examples of methods used for solving MILP problems are *branch-and-bound* [29] and *general cutting planes method* [30]. A branch-and-bound works as follows: by removing the integrality restrictions from the original MILP formulation, so-called linear programming *relaxation* is obtained. If its optimal solution satisfies all of the original integrality restrictions, then it is also an optimal solution to the original MILP problem. Otherwise, branching is performed on some variable x, which should be integral in the original MILP problem, but is fractional in the relaxed solution x^*. Usually, two branches are created, imposing constraints $x \leq \lfloor x^* \rfloor$ and $x \geq \lceil x^* \rceil$, respectively. This way, two more-restricted MILP are created; if both are solved, then the better of the two solutions is also an optimal solution to the original problem. This branching idea is applied recursively; Figure 2.4 provides an illustration of the procedure. The best-so-far integer solution found during branching is usually called an *incumbent* solution. Its objective value can be used to prune branches whose optimal relaxed objective value is worse.

General cutting planes method is also trying to eliminate fractional solutions, thus improving the relaxation. However, instead of creating new sub-problems (like in the branch-and-bound), it is done during the solution process itself. The idea is that linear inequality, called *cut*, is added to separate the fractional optimum of the relaxed problem from the convex hull of the true feasible set. An illustration is provided in Figure 2.5.

In practice, a branch-and-bound procedure is often integrated with a cutting planes method, together forming so-called *branch-and-cut* method – the cutting planes are used to tighten the linear relaxations during the branch-and-bound procedure.

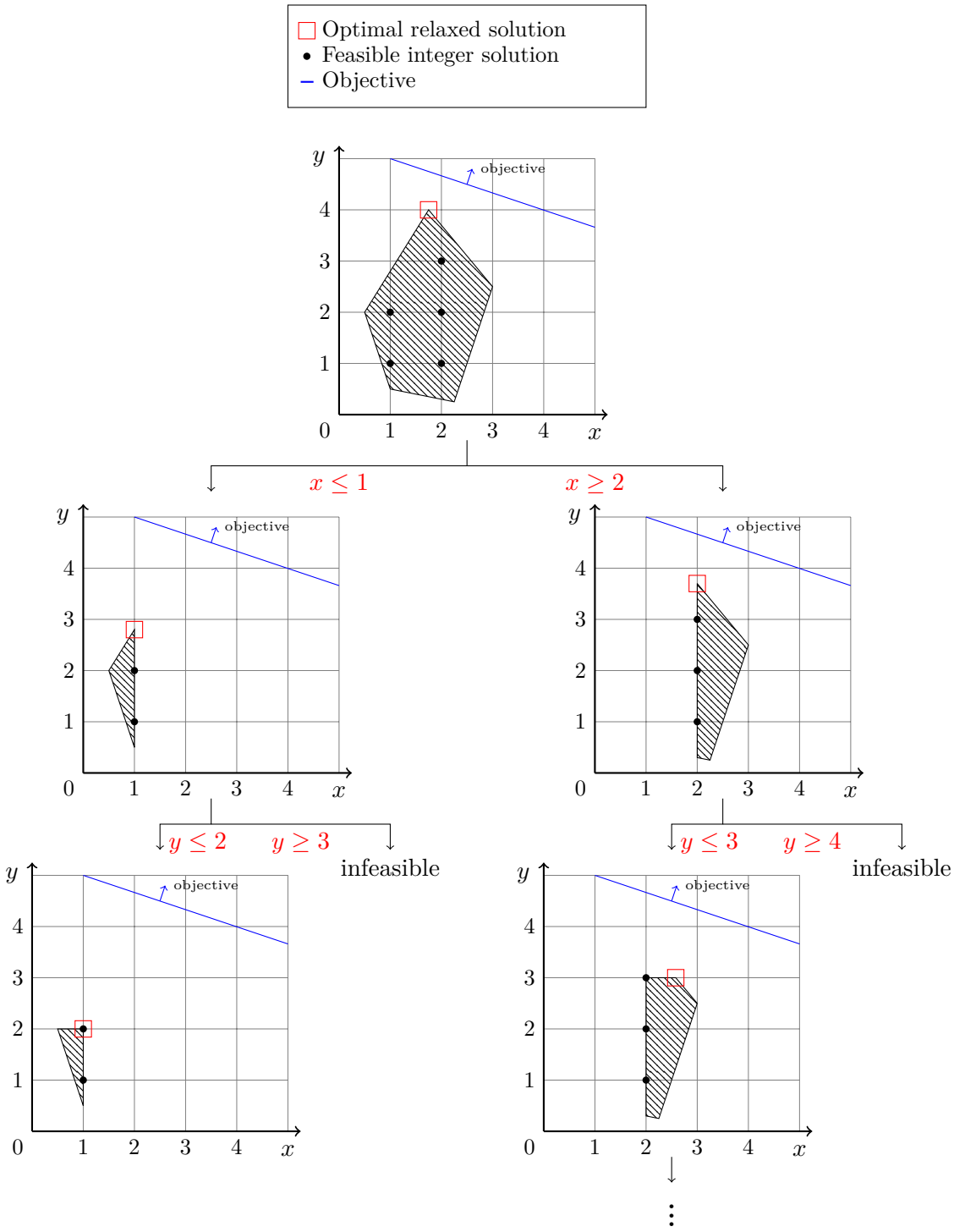


Figure 2.4: Illustration of a branching procedure for MILP solving

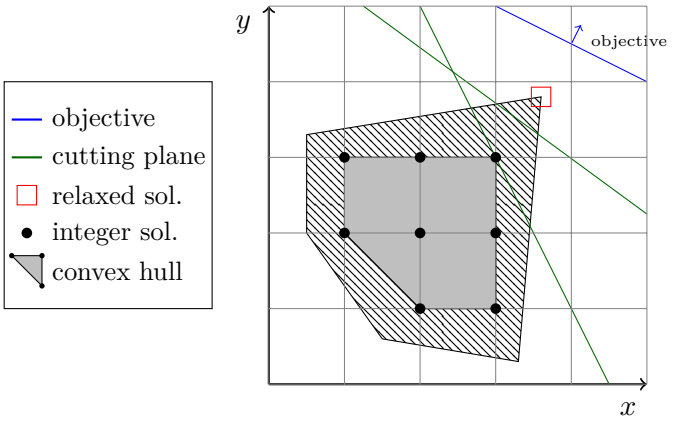


Figure 2.5: An example of two possible cutting planes separating the fractional solution of the relaxed problem from the convex hull of the MILP feasible solutions

2.3.2 Dantzig-Wolfe decomposition

Various decomposition/reformulation approaches are often used on integer programs to obtain strong(er) relaxation or to reduce symmetries. Dynamic addition of variables (i.e. columns) or constraints (cutting planes) is usually part of these approaches. Dantzig and Wolfe [22] studied structured linear problems, which they decomposed into smaller linear programs, whose solutions were obtained through a generalized simplex method. Their work was pioneering (core ideas were published in the 1960s) and strongly influenced the whole field of study. Some of their ideas will be briefly described in the following paragraphs.

Original problem

Let us assume a linear program (2.5) with a structure illustrated in Figure 2.6. Its matrix is block-angular and consists of *connecting* constraints and *independent* constraints. Connecting constraints bind the columns together whereas independent constraints describe individual sub-problems. In the area of parallel-machine scheduling, independent constraints could describe the behaviour of individual machines, while connecting constraints would, for example, limit the number of machines allowed to operate simultaneously.

$$\begin{aligned} \underset{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n}{\text{minimize}} \quad & \sum_{j=1}^{n} \boldsymbol{c}_j^T \boldsymbol{x}_j \\ \text{subject to} \quad & \sum_{j=1}^{n} \boldsymbol{A}_j \boldsymbol{x}_j \geq \boldsymbol{b}, \\ & \boldsymbol{B}_j \boldsymbol{x}_j \geq \boldsymbol{b}_j, \quad \forall j \in \{1,\ldots,n\} \\ & \boldsymbol{x}_j \geq \boldsymbol{0}, \qquad \forall j \in \{1,\ldots,n\}, \end{aligned} \tag{2.5}$$

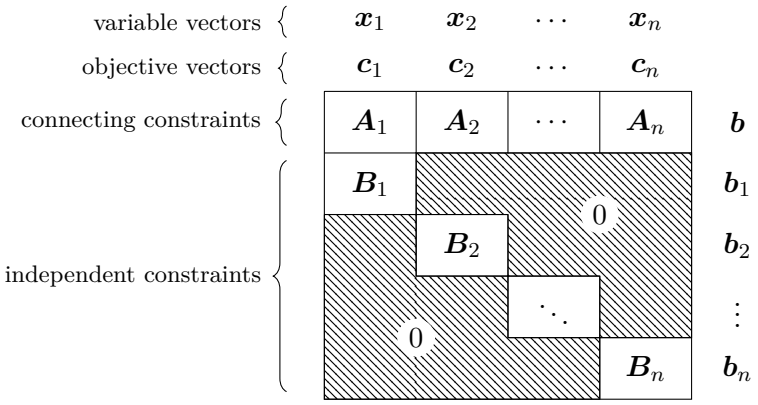


Figure 2.6: An illustration of desired matrix structure for the Dantzig-Wolfe decomposition

Dantzig and Wolfe followed the idea, that independent constraints are altogether of secondary importance and should be studied mainly through the restrictions they impose on the connecting constraints. For the simplicity, let us assume that polyhedron

$$S_j = \{\boldsymbol{x}_j \mid \boldsymbol{B}_j\boldsymbol{x}_j \geq \boldsymbol{b},\ \boldsymbol{x}_j \geq \boldsymbol{0}\} \tag{2.6}$$

is bounded (i.e. polytope) for each j. The general case is further discussed in [31]. Now, the *original problem* (2.5) can be reformulated to a *master problem*.

Master problem

Let us denote a set of all extreme points of the convex polyhedron S_j by

$$W_j = \{\boldsymbol{x}_{j,1}, \ldots, \boldsymbol{x}_{j,K_j}\}. \tag{2.7}$$

Let us also define substitutions

$$\boldsymbol{p}_{j,k} = \boldsymbol{A}_j\boldsymbol{x}_{j,k}, \tag{2.8}$$

$$c_{j,k} = \boldsymbol{c}_j^T\boldsymbol{x}_{j,k}, \tag{2.9}$$

for all j in $\{1, \ldots, n\}$ and k in $\{1, \ldots, K_j\}$. Now, *the extremal program* (also called master problem) can be formulated as (2.10).

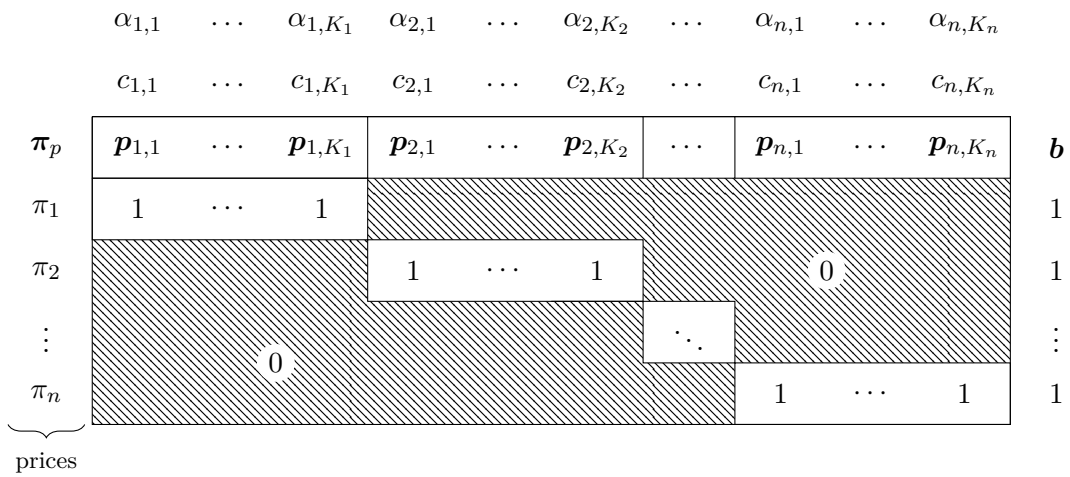


Figure 2.7: Structure of the master problem

$$
\begin{aligned}
\underset{\boldsymbol{\alpha}}{\text{minimize}} \quad & \sum_{j,k} c_{j,k} \cdot \alpha_{j,k} \\
\text{subject to} \quad & \sum_{j,k} \boldsymbol{p}_{j,k} \cdot \alpha_{j,k} \geq \boldsymbol{b}, \\
& \sum_{k=1}^{K_j} \alpha_{j,k} = 1, \qquad \forall j \in \{1, \ldots, n\}, \\
& \alpha_{j,k} \geq 0, \qquad \forall j \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, K_j\},
\end{aligned}
\tag{2.10}
$$

where variables $(\alpha_{1,1}, \ldots, \alpha_{1,K_1}, \ldots, \alpha_{n,1}, \ldots, \alpha_{n,K_n}) = \boldsymbol{\alpha}$ are in fact the coefficients of convex combinations of an individual points of W_j. Structure of the problem can be seen in Figure 2.7. Dantzig and Wolfe pointed out, that if a solution of (2.10) could be obtained, it could be simply transformed to a solution of the original problem (2.5). Indeed, any point $\boldsymbol{x}_j$ of S_j (bounded convex polyhedral set) can be written as a convex combination of its extreme points [32], i.e., $\sum_k \boldsymbol{x}_{j,k} \cdot \alpha_{j,k}$, where $\sum_k \alpha_{j,k} = 1, \alpha_{j,k} \geq 0 \ \forall j, k$. Expanding the objective and the first set of constraints of (2.10), we get

$$
\sum_j \sum_k c_{j,k} \cdot \alpha_{j,k} = \sum_j \sum_k \boldsymbol{c}_j^T \cdot \boldsymbol{x}_{j,k} \cdot \alpha_{j,k} = \sum_j \boldsymbol{c}_j^T \underbrace{\sum_k (\boldsymbol{x}_{j,k} \cdot \alpha_{j,k})}_{\text{some } \boldsymbol{x}_j \in S_j}, \tag{2.11}
$$

$$
\sum_j \sum_k \boldsymbol{p}_{j,k} \cdot \alpha_{j,k} = \sum_j \sum_k \boldsymbol{A}_j \boldsymbol{x}_{j,k} \cdot \alpha_{j,k} = \sum_j \boldsymbol{A}_j \underbrace{\sum_k (\boldsymbol{x}_{j,k} \cdot \alpha_{j,k})}_{\text{some } \boldsymbol{x}_j \in S_j}. \tag{2.12}
$$

Now, it is easy to see, that problems (2.5) and (2.10) are nearly the same. Finding the optimal vector $\boldsymbol{x}_j$ can be done by finding the convex combination coefficients in the respective

polyhedron S_j. The only practical difference is that the numbers of variables/constraints differ. The connecting constraints are present in both models, but the independent constraints of the original formulation (2.5) were replaced by n constraints of type $\sum_k \alpha_{j,k} = 1$. This could lead to a considerable reduction if the number of independent constraints was large compared to the number of individual sub-problems n. However, the number of variables increased significantly as the number of extreme points of a polyhedron is, in general, exponential with respect to the input size of the problem. Therefore, the discussed decomposition would not help much if it was not possible to reduce the number of variables of the master problem. Fortunately, the reduction can be achieved by a *column generation* technique.

Column generation

To deal with a large number of columns of the master problem, the idea is to include only their subset R, thus forming a *restricted master problem* (RMP). Columns are generated iteratively by solving *pricing problems*. To decide, whether a column should be included or not, dual variables $\boldsymbol{\pi} = (\boldsymbol{\pi}_p, \pi_1, \ldots, \pi_n)$ of the current solution are used. *Reduced cost* of column (j,k) is defined as

$$c_{j,k} - \boldsymbol{\pi}_p^T \boldsymbol{p}_{j,k} - \pi_j, \tag{2.13}$$

which is a difference of coefficient $c_{j,k}$ and a scalar product of the price vector and the (j,k)-th column. Expanding the expression (2.13), we get

$$\begin{aligned} c_{j,k} - \boldsymbol{\pi}_p^T \boldsymbol{p}_{j,k} - \pi_j &= \boldsymbol{c}_j^T \boldsymbol{x}_{j,k} - \boldsymbol{\pi}_p^T \boldsymbol{A}_j \boldsymbol{x}_{j,k} - \pi_j \\ &= \left(\boldsymbol{c}_j^T - \boldsymbol{\pi}_p^T \boldsymbol{A}_j\right) \boldsymbol{x}_{j,k} - \pi_j, \end{aligned} \tag{2.14}$$

therefore, the pricing problem can be stated as follows

$$\min \left\{ \left(\boldsymbol{c}_j^T - \boldsymbol{\pi}_p^T \boldsymbol{A}_j\right) \boldsymbol{x}_j - \pi_j \mid \boldsymbol{x}_j \in S_j \right\}, \; j = 1, \ldots, n. \tag{2.15}$$

For a minimization problem, a column has a potential to improve the current solution of the restricted master problem, if its reduced cost is negative [31]. As there are n individual sub-problems, pricing problem (2.15) is solved for each of them separately. If the minimum is negative (and finite), the optimal solution is an extreme point and a new variable with a column corresponding to this optimal extreme point is added to the restricted master problem. An optimal solution to the restricted master problem is found when no minimum in (2.15) is negative.

Note that for simplicity, we assumed that S_j are bounded. For an unbounded case, it may happen that solution to the pricing problem would be minus infinity. This can be handled, but the formulation would become slightly (but not much) more complicated; see, e.g., [33].

In this section, the decomposition was used on the linear programs, however, in combination with a branch-and-bound procedure it can be used to solve the integer programs too. The integration of a column-generation and branch-and-bound is often called branch-and-price (referencing the pricing problems) and is described in Section 2.3.4.

2.3.3 Lagrangian relaxation

In the last section, Dantzig-Wolfe decomposition technique was used to simplify the original model. It keeps connecting constraints in the master problem while exploiting the structure in the sub-problems. On the other hand, *Lagrangian relaxation* can be used to relax connecting constraints $\sum_j \boldsymbol{A}_j \boldsymbol{x}_j \geq \boldsymbol{b}$ by penalizing their violation in the objective function. Let us simplify the notation by rewriting $\sum_j \boldsymbol{A}_j \boldsymbol{x}_j \geq \boldsymbol{b}$ simply to $\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}$ and similarly $\sum_j \boldsymbol{c}_j^T \boldsymbol{x}_j$ to $\boldsymbol{c}^T \boldsymbol{x}$. Now, the *Lagrangian sub-problem* can be written as

$$L(\boldsymbol{\lambda}) = \min_{\boldsymbol{x}} \boldsymbol{c}^T \boldsymbol{x} - \boldsymbol{\lambda}^T (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}), \tag{2.16}$$

where $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n),\ \boldsymbol{x}_j \in S_j, \forall j \in \{1, \ldots, n\}$ and $\boldsymbol{\lambda} \geq \boldsymbol{0}$. Vector $\boldsymbol{\lambda}$ penalizes violations of relaxed constraints. For fixed $\bar{\boldsymbol{\lambda}}$, Lagrangian sub-problem gives a lower bound to the original problem (2.5), i.e.,

$$\boldsymbol{c}^T \bar{\boldsymbol{x}} \geq \boldsymbol{c}^T \bar{\boldsymbol{x}} - \bar{\boldsymbol{\lambda}}^T (\boldsymbol{A}\bar{\boldsymbol{x}} - \boldsymbol{b}) \geq \boldsymbol{c}^T \hat{\boldsymbol{x}} - \bar{\boldsymbol{\lambda}}^T (\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{b}), \tag{2.17}$$

where $\bar{\boldsymbol{x}}$ is the optimal solution to the original problem and $\hat{\boldsymbol{x}}$ is the optimal solution to the Lagrangian relaxation (2.16). The first inequality holds because $\bar{\boldsymbol{x}}$ is feasible to the original problem ($\boldsymbol{A}\bar{\boldsymbol{x}} - \boldsymbol{b} \geq 0$) and the second one holds because $\hat{\boldsymbol{x}}$ is the optimal solution to the Lagrangian sub-problem.

Seeing that Lagrangian sub-problem provides a lower bound to the original problem, one step further can be done – the lower bound can be tightened as much as possible by solving a *Lagrangian dual problem*

$$\max_{\boldsymbol{\lambda}} L(\boldsymbol{\lambda}),\ \text{s.t. } \boldsymbol{\lambda} \geq \boldsymbol{0}. \tag{2.18}$$

It has been proven, that Lagrangian relaxation and Dantzig-Wolfe decomposition are tightly connected together. In fact, when relaxing exactly the linking constraints of the original problem, the optimal values to the Lagrangian dual and LP relaxation of the

Dantzig-Wolfe decomposition are the same and one formulation is the dual of the other one [34]. Moreover, the optimal dual variables $\boldsymbol{\pi}$ for the linking constraints in the master problem correspond to the optimal multipliers $\boldsymbol{\lambda}$ [35]. While in the column generation, the values of $\boldsymbol{\pi}$ are obtained by solving the LP relaxation of the RMP, in the Lagrangian relaxation, the multipliers are usually updated by sub-gradient methods [36]. Both approaches have advantages and disadvantages; some of them are mentioned in [36].

Even though the Lagrangian relaxation is not used in this work, it provides a different perspective and solution approach; therefore it was briefly mentioned to widen the context and show the connections.

2.3.4 Branch and price

In this section, several comments on main parts of the branch-and-price algorithm are given. Further details can be found in [19, 23, 33].

Branch-and-price is an algorithm which integrates the branch-and-bound procedure and column generation techniques to solve large-scale/hard integer programs. At first, the original problem needs to be decomposed into a master problem and a pricing problem. At each node of the branch and bound tree, the restricted master problem is repeatedly solved. Individual columns, identified by the pricing problem, are iteratively added to RMP to ensure the optimality. If there is no column to add, integrality of the solution is tested. If the solution is not integral, branching occurs and the process is repeated. The whole procedure is illustrated in Figure 2.8.

Pricing

One of the important parts of the algorithm is a column generation, i.e., iterative solving of the pricing problem and addition of the columns.

As described in Section 2.3.2, columns with negative reduced cost are iteratively generated. The pricing problem, which is repeatedly solved, is problem specific and even though it is simpler than the original problem (as it is only a single part of the whole decomposition), it can still be hard to solve. Therefore, there is a reasonable question asking *how to speed-up solving of the pricing problem.* One solution is straightforward – as any column with a negative reduce cost has a potential to improve the current solution, the first one can be used; meaning that there is no need to solve the pricing problem optimally. Heuristics can be used to solve the pricing problem in a fast way and only if they do not find any solution with a negative reduced cost, the optimal solver needs to be called.

To improve the performance even further, one can try to predict next reduced cost by a prediction model trained on the previous iterations; the prediction provides an assumed lower bound, which can speed-up the solving process of the pricing problem. Again, if no

solution with a negative reduced cost is found, the optimal solver needs to be called. This method was developed in [37] and authors claim significant improvements, up to 40 % of total solving time.

Branching

Another crucial part of the branch-and-price design is an integration of the branch-and-bound procedure and a selection of the branching scheme.

A naive approach would be to select one of the fractional variables of the RMP and to create two branches by rounding the solution up and down, respectively.; see Figure 2.4. This, however, may not be the best choice, as the variables of the RMP have only weak connections to the variables of the original problem and branching could lead to significant changes of the pricing problem. Moreover, the branching tree could become unbalanced sometimes [19]. Therefore other branching strategies, like those described in [18, 19, 38], are often considered to improve the overall performance of the algorithm. Note, that similarly to a branch-and-bound, some branches can be pruned out by the best-so-far integer solution (found at the end of the column generation phase in some node), which provides an upper bound to the solution.

Initialization

Usually, there are two problematic situations arising during the solution process, when the RMP does not need to be feasible (even though the master problem is); the first one is in the beginning, when no columns are generated, and the second one is after branching, when some columns may be deleted due to the restrictions imposed by the branching scheme. Heuristics can be used to find the initial set of columns. A good heuristic may have a significant effect on the total solution time; however, it does not guarantee the initial feasibility in general. One possible method how to deal with the problem is to introduce artificial variables (relaxing the constraints of the RMP) with "big M" penalty costs [39]. There are, of course, other possibilities, such as the Farkas pricing method [33].

Further reading

The branch-and-price is a popular method to solve large MILP nowadays and so there are many publications about the method. Authors describe various stabilization techniques [33, 40, 41] as well as problem-specific improvements, e.g., [19, 42, 43]. A brief introduction to various topics associated with a branch-and-price is provided, e.g., by [19, 33].

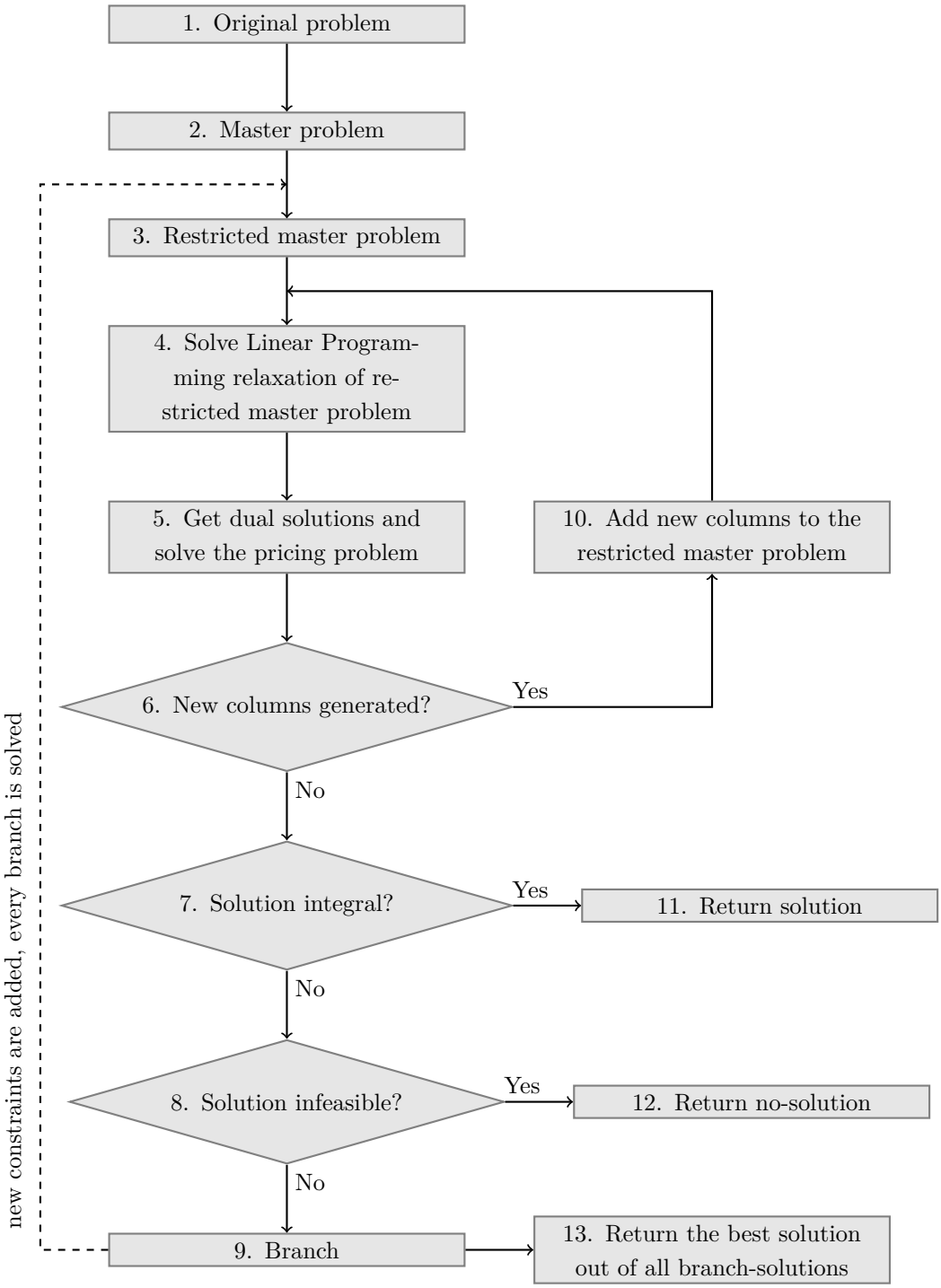


Figure 2.8: Diagram of a branch-and-price algorithm

2.4 Constraint programming

Constraint programming (CP) is an optimization technique used to find solutions to combinatorial optimization problems, such as scheduling problems, which may have numerous feasible solutions. It has some similarities to mathematical programming approaches (LP, MILP), but it utilizes different optimization approaches. Contrary to LP, which has a strong theoretical ground in algebra, CP originated in a field of artificial intelligence and strongly relies on graph theory and various search and propagation techniques.

Differences between CP and MILP

A CP model is described declaratively, using decision variables, constraints and an objective, which is minimized or maximized; that is similar to MILP. However, CP uses only discrete decision variables (boolean or integral), whereas mathematical programming models can combine discrete and continuous variables. Constraints of a CP model can have various forms – there are basically no limitations, as long as the propagation is fast. CP has native support of non-linear costs and constraints, logical constraints, global constraints (such as *allDifferent* constraint) and problem specific constraints. That is a huge difference to LP and MILP, whose models can be constrained only by linear constraints.

Solution approach

Solution to a CP model is obtained constructively; values are assigned to variables, expanding partial solutions to a complete solution. As soon as some constraint is violated, it is useless to expand further; a backtracking procedure is employed to try other assignments. This way, it is possible to try all of the assignments; however, that would be highly inefficient. In order to reduce the search-space, various domain filtering techniques are used. Once a first solution is found, the search proceeds to find further solutions with better objective values.

References

Basics of CP and multiple search/propagation techniques can be found in the lectures of Roman Barták [44]. A reader can find out more about applications, modelling concepts and examples of state of the art IBM ILOG CP Optimizer in [45].

3 Models

In this chapter, individual models for the problem defined in Section 2.1 will be described. In Section 3.1, the whole problem will be written as a single complex model, here called the *global model*. Two formulation approaches (MILP and CP) will be used for comparison. Afterwards, in Section 3.2, the global MILP model will be decomposed to a master problem and a pricing problem. Again, MILP and CP models of the pricing problem will be created. Finally, in Section 3.3, a reference model adapted from the literature will be established in order to compare approaches proposed in this work with a state of the art model.

Note that the machine profiles can have different lengths – up to $i_{\max}$, but it is not desirable to optimize over them all – in case of the global model, it would mean to try all possible combinations of modes over all different lengths up to $i_{\max}$. That is surely not efficient. To avoid that, a simple transformation of the transition graph can be done such that only the profiles of length $i_{\max}$ will be considered. The trick is to add a *dummy mode* with zero power consumption, $\mathrm{t}^{\min}_{\mathrm{dummy}} = 0$, $\mathrm{t}^{\max}_{\mathrm{dummy}} = \mathrm{h}$, connected to the terminal mode and itself. The transitions will have a zero length and cost; therefore the dummy mode will not affect the objective. As there won't be any outgoing edge (except for the reflexive edge), a machine would not be able to change its mode once entering the dummy mode. Thus, the dummy mode will appear at the end of a profile (if scheduled). Optimal profiles with the dummy mode would correspond to the profiles, whose length was smaller than $i_{\max}$ for the original transition graph. For illustration, the transformation of the example graph from Section 2.1 is shown in Figure 3.1. A similar effect would be achieved by adding only the reflexive edge to the terminal state, however, this way it would be simpler to add the additional restrictions, such that some mode can appear only x-times in the profile of a machine etc.

In the following sections, the term *transition graph* stands for the transition graph after the transformation if not stated otherwise.

3.1 Global models

Two models, representing two different formulation approaches, will be described in this section. Both models will capture the whole problem, including all of the restrictions. The first formulation will be based on MILP while the second one will be written as a CP problem. Both of the frameworks (MILP, CP) are widely used [3]; mainly because they can describe many real-life problems as it is usually easy to formulate all kinds of

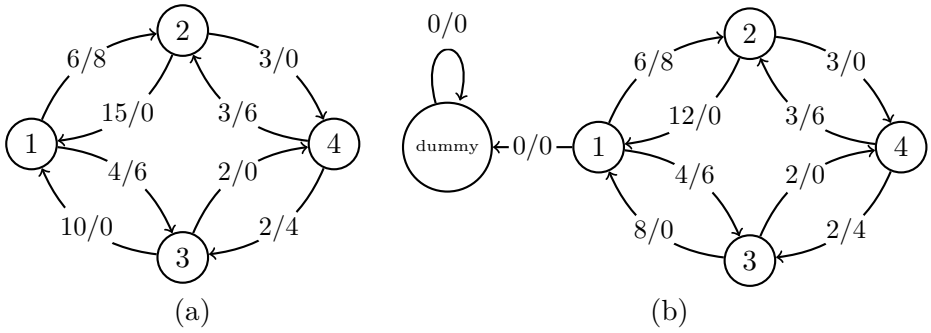


Figure 3.1: Example of a transition graph transformation: (a) the original graph, (b) the transformed graph

constraints. In combination with the state of the art solvers, such as Gurobi, CPLEX or ILOG CP solver, they are very powerful (and often easy to use) and should be considered as the first solution approach.

The main idea behind the formulations is to explicitly model *intervals* i, which represent the individual modes – together forming a profile of each machine. Each interval will be characterized by its start time and length, and the individual intervals will be linked together by a set of constraints so that they form a feasible profile (starting with v_{init}, ending with v_{term}, satisfying restrictions given by the transition graph, ...). Jobs will be characterized by their start time and their assignment to a machine. A set of constraints will link a job with its assigned machine such that the assignment is feasible (satisfying all of the requirements defined in Section 2.1).

An assignment, together with a machine profile, operating times and start time of the job, directly influence the processing time of the job – this is the hard part because the processing time depends on the mode of a machine, which is determined by the profile and operating times, but those are optimized. In general, the processing time of the job in the optimal schedule is not known a priory.

3.1.1 MILP model

Two basic types of variables (integer and continuous) are used in MILP models. They are linked together by a set of linear inequalities (constraints). A linear objective is formed to specify qualities of the solution that are being looked for.

A global MILP model will be described in the following way – firstly, used variables will be listed, and their meaning will be explained; secondly, individual constraints will be formed, reflecting the restrictions of the problem; at last, the optimization objective will be given, concluding the description.

Variables

There are four types of binary variables used in the model: $x_{m,i,v}$, $z_{m,i,v,v'}$, $y_{j,m,i,v}$ and $a_{j,j'}$; other two types of variables, s_j and $p_{m,i,v}$, are integral. An interpretation of the individual variables is following:

$$x_{m,i,v} = \begin{cases} 1 & \text{if machine } m \text{ operates in mode } v \text{ during interval } i, \\ 0 & \text{otherwise,} \end{cases}$$

$$z_{m,i,v,v'} = \begin{cases} 1 & \text{if machine } m \text{ changes its mode from } v \text{ to } v' \text{ between} \\ & \text{intervals } i \text{ and } (i+1), \\ 0 & \text{otherwise,} \end{cases}$$

$$y_{j,m,i,v} = \begin{cases} 1 & \text{if job } j \text{ is processed by machine } m \text{ in mode } v \text{ during} \\ & \text{interval } i, \\ 0 & \text{otherwise,} \end{cases}$$

$$a_{j,j'} = \begin{cases} 1 & \text{if job } j \text{ is processed before job } j' \text{ (if both are assigned} \\ & \text{to the same machine),} \\ 0 & \text{otherwise,} \end{cases}$$

s_j ... start time of job j,

$p_{m,i,v}$ ... time, which machine m spent in mode v during interval i.

Variables s_j and $p_{m,i,v}$ could also be defined as continuous variables, but it is assumed that the scheduling horizon is discretized – it allows scheduling by minutes, hours or even days (depending on the application) and it also prevents some of the numerical issues.

Besides the variables defined above, an alias $s_{m,i}$ is used to represent the start time of i-th interval on machine m. It can be expressed as

$$\forall m : s_{m,i} = \begin{cases} 0 & \text{if } i = 1, \\ s_{m,i-1} + \sum_{v=1}^{V} p_{m,i-1,v} + \sum_{v=1}^{V} \sum_{v'=1}^{V} z_{m,i-1,v,v'} \cdot \mathrm{t}_{v,v'} & \text{if } i > 1. \end{cases} \tag{3.1}$$

The recursive definition (3.1) states that the first interval starts at time 0 and the i-th interval starts just after the transition from the previous one is finished (summing the start time and processing time of the $(i-1)$-th interval together with the corresponding transition time). The constraints (3.4) to (3.6) will ensure that $p_{m,i,v}$ will have at most one non-zero value over v for all m and i. Similarly, only one transition between modes will be possible for given i, m, therefore only the appropriate transition time will be added. The expression $s_{m,i}$ is linear and its unexpanded form will be used to simplify the notation.

Constraints

At first, let's describe the machines and their modes. The schedule needs to start with the initial mode v_{init} and end with the terminal mode v_{term} or dummy mode v_{dummy} for each machine m. These can be written simply as (3.2) and (3.3), respectively. Of course, exactly one mode at a time has to be active, which leads us to (3.4). Time spent in each mode $v \in \mathcal{V}$ is restricted by $\mathrm{t}_v^{\min}$ (3.5) and $\mathrm{t}_v^{\max}$ (3.6) and the last interval has to end at h (3.7).

$$x_{m,1,v_{\text{init}}} = 1, \qquad \forall m \in \mathcal{M}, \tag{3.2}$$

$$x_{m,i_{\max},v_{\text{term}}} + x_{m,i_{\max},v_{\text{dummy}}} = 1, \qquad \forall m \in \mathcal{M}, \tag{3.3}$$

$$\sum_{v \in \mathcal{V}} x_{m,i,v} = 1, \qquad \forall m \in \mathcal{M}, \forall i \in \mathcal{I}, \tag{3.4}$$

$$\mathrm{t}_v^{\min} \cdot x_{m,i,v} \leq p_{m,i,v}, \qquad \forall m \in \mathcal{M}, \forall i \in \mathcal{I}, \forall v \in \mathcal{V}, \tag{3.5}$$

$$\mathrm{t}_v^{\max} \cdot x_{m,i,v} \geq p_{m,i,v}, \qquad \forall m \in \mathcal{M}, \forall i \in \mathcal{I}, \forall v \in \mathcal{V}, \tag{3.6}$$

$$s_{m,i_{\max}} + \sum_{v \in \mathcal{V}} p_{m,i_{\max},v} = \mathrm{h}, \qquad \forall m \in \mathcal{M}. \tag{3.7}$$

Note, that constraints (3.5) and (3.6) force $p_{m,i,v}$ to zero if $x_{m,i,v} = 0$, which is for all but one mode for each $m \in \mathcal{M}, v \in \mathcal{V}$. Therefore, $\sum_v p_{m,i,v}$ gives exactly the time, which machine m spends on interval i.

Only valid transitions between modes can occur in the schedule, so if machine m operates in mode v during interval i then it has to operate in mode v' during interval $(i+1)$ such that $v' \in \text{Next}(v) = \{v' \in \mathcal{V} \mid \exists e \in \mathcal{E} \text{ s.t. } e = (v, v')\}$. Similarly, it can be written as $v \in \text{Prev}(v') = \{v \in \mathcal{V} \mid \exists e \in \mathcal{E} \text{ s.t. } e = (v, v')\}$. It means that two modes can follow one after other only if there is a transition between these modes in the transition graph. In a form of constraint, it can be expressed as (3.8) or (3.9), respectively.

What remains is to link variables $x_{m,i,v}$ with variables $z_{m,i,v,v'}$. It can be done by two types of constraints: one, (3.10), makes $z_{m,i,v,v'}$ equal to 1 if there are modes v and v' scheduled on machine m during interval i and $(i+1)$, respectively. The other one, (3.11), makes sure that $z_{m,i,v,v'}$ is zero otherwise, i.e., only one $z_{m,i,\cdot,\cdot}$ is set to 1 for each machine m and interval i.

$$x_{m,i,v} \leq \sum_{v' \in \mathrm{Next}(v)} x_{m,i+1,v'}, \quad \forall m \in \mathcal{M}, i \in \{1, \ldots, i_{\max} - 1\}, \tag{3.8}$$

$$x_{m,i,v'} \leq \sum_{v \in \mathrm{Prev}(v')} x_{m,i-1,v}, \quad \forall m \in \mathcal{M}, i \in \{2, \ldots, i_{\max}\}, \tag{3.9}$$

$$x_{m,i,v} + x_{m,i+1,v'} - 1 \leq z_{m,i,v,v'}, \quad \forall m \in \mathcal{M}, \forall i \in \{1, \ldots, i_{\max} - 1\}, \forall v, v' \in \mathcal{V}, \tag{3.10}$$

$$\sum_{v \in \mathcal{V}} \sum_{v' \in \mathcal{V}} z_{m,i,v,v'} \leq 1, \quad \forall m \in \mathcal{M}, \forall i \in \{1, \ldots, i_{\max} - 1\}. \tag{3.11}$$

Now, machines were described, and only the integration of the jobs remains. Simple constraints like that each job has to be scheduled, (3.12), has to start after its release time, (3.13) and has to end before its deadline, (3.14), are easy to write. Note, that by capturing both the mode and the interval by variables $y_{j,m,i,v}$, the processing time of job j can be expressed by $\sum_{m \in \mathcal{M}} \sum_{i \in \mathcal{I}} \sum_{v \in \mathcal{V}} (y_{j,m,i,v} \cdot \mathrm{p}_{j,v})$ as the only one of these variables will be active (equal to one) in a feasible solution.

$$\sum_{m \in \mathcal{M}} \sum_{i \in \mathcal{I}} \sum_{v \in \mathcal{V}} y_{j,m,i,v} = 1, \quad \forall j \in \mathcal{J}, \tag{3.12}$$

$$s_j \geq \mathrm{r}_j, \quad \forall j \in \mathcal{J}, \tag{3.13}$$

$$s_j + \sum_{m \in \mathcal{M}} \sum_{i \in \mathcal{I}} \sum_{v \in \mathcal{V}} (y_{j,m,i,v} \cdot \mathrm{p}_{j,v}) \leq \mathrm{d}_j, \quad \forall j \in \mathcal{J}. \tag{3.14}$$

Slightly more difficult is to link the start time of a job with its assigned interval. A large constant, here denoted by K, needs to be used. If job j is scheduled onto interval i on machine m, then s_j needs to be greater than or equal to $s_{m,i}$, (3.15); otherwise, the two start times do not have a connection and the respective constraint has to be inactive, which is achieved by using constant K. Similarly, a link between a job and a deadline of the associated interval has to be formed (3.16). In practice, it is not possible to set $\mathrm{K} = \infty$; it has to be chosen carefully with respect to the constraint in which it is used. Constant K_1 has to be set at least to h in order for the constraints (3.15) to work. Similarly $\mathrm{K}_2 \geq 2 \cdot \mathrm{h}$.

$$s_{m,i} \leq s_j + \mathrm{K}_1 \cdot \left(1 - \sum_{v \in \mathcal{V}} y_{j,m,i,v}\right), \qquad \forall m \in \mathcal{M}, i \in \mathcal{I}, j \in \mathcal{J}, \quad (3.15)$$

$$s_j + y_{j,m,i,v} \cdot \mathrm{p}_{j,v} \leq s_{m,i} + p_{m,i,v} + \mathrm{K}_2 \left(1 - y_{j,m,i,v}\right), \quad \forall m \in \mathcal{M}, i \in \mathcal{I}, j \in \mathcal{J}, v \in \mathcal{V}. \quad (3.16)$$

The final problem to be solved is the overlapping of the tasks scheduled on the same machine and interval. Using variables $a_{j,j'}$, the constraints saying that *one job has to end before the other one starts* can be formulated as (3.17). Again, big constant K_3 is used to deactivate the constraints if the two jobs are not scheduled to the same machine and interval or if the precedence is not active ($a_{j,j'} = 0$). Constraint (3.18) just states that either job j should be processed before job j' or the other way around.

There are many constraints of type (3.17); for some of the bigger models, it would not even be feasible to build them in a reasonable time. However, not all of the constraints are active in the optimal solution. Therefore, it is possible to generate them *lazily* during the solution process (i.e. using them as cuts).

$$s_j + \sum_{v \in \mathcal{V}} y_{j,m,i,v} \cdot \mathrm{p}_{j,v} \leq s_{j'} + \mathrm{K}_3 \cdot \left(3 - \sum_{v \in \mathcal{V}} y_{j,m,i,v} - \sum_{v \in \mathcal{V}} y_{j',m,i,v} - a_{j,j'}\right), \quad (3.17)$$

$$a_{j,j'} = 1 - a_{j',j}, \quad \forall m \in \mathcal{M}, i \in \mathcal{I}, \forall j, j' \in \mathcal{J}, j \neq j' \quad (3.18)$$

Objective

The objective (2.2), reformulated using the variables established in this section, can be written as (3.19). It is linear and can be optimized by any standard MILP solver, such as Gurobi or CPLEX Optimizer.

$$\underbrace{\sum_{m \in \mathcal{M}} \sum_{i \in \mathcal{I}} \sum_{v \in \mathcal{V}} p_{m,i,v} \cdot \mathrm{w}_v}_{\text{operating cost}} + \underbrace{\sum_{m \in \mathcal{M}} \sum_{i \in \{1,\ldots,i_{\max}-1\}} \sum_{v \in \mathcal{V}} \sum_{v' \in \mathcal{V}} z_{m,i,v,v'} \cdot \mathrm{c}_{v,v'}}_{\text{transition cost}} \quad (3.19)$$

Additional constraints

In order to help solvers, additional constraints can be added, providing further information about the variables and their connections, thus potentially improving the solution time

of the model. Constraints (3.20) are such an example. They state that for each machine m the time spent in mode v during interval i has to be longer than or equal to the time spent on all of the jobs, which were assigned to be processed in mode v during interval i on this machine. It is very trivial; however, it leads to a significant improvement.

$$\sum_{j \in \mathcal{J}} \left(y_{j,m,i,v} \cdot \mathrm{p}_{j,v} \right) \quad \leq \quad p_{m,i,v}, \quad \forall m \in \mathcal{M}, i \in \mathcal{I}, v \in \mathcal{V}. \tag{3.20}$$

As the machines are identical, there are multiple solutions with the same quality, which differ only in the permutation of the machines. The specific order is not important, so additional constraints, imposing some kind of ordering, can be used to reduce these symmetries. For example, machines can be ordered by non-increasing total time spent on the processing of the jobs, (3.21).

$$\sum_{j \in \mathcal{J}} \sum_{i \in \mathcal{I}} \sum_{v \in \mathcal{V}} \left(y_{j,m,i,v} \cdot \mathrm{p}_{j,v} \right) \quad \geq \quad \sum_{j \in \mathcal{J}} \sum_{i \in \mathcal{I}} \sum_{v \in \mathcal{V}} \left(y_{j,m+1,i,v} \cdot \mathrm{p}_{j,v} \right), \quad \forall m \in \{1, \ldots, M-1\}. \tag{3.21}$$

Complete model

Finally, the objective and the constraints described above can be joined to form a single MILP model (3.22).

$$
\begin{aligned}
\text{minimize} \sum_{m\in\mathcal{M}}\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} p_{m,i,v}\cdot \mathrm{w}_v + \sum_{m\in\mathcal{M}}\sum_{i\in\{1,\ldots,i_{\max}-1\}}\sum_{v\in\mathcal{V}}\sum_{v'\in\mathcal{V}} z_{m,i,v,v'}\cdot \mathrm{c}_{v,v'} \quad \text{s.t.}\\
x_{m,1,v_{\text{init}}} &= 1, && \forall m\in\mathcal{M},\\
x_{m,i_{\max},v_{\text{term}}} + x_{m,i_{\max},v_{\text{dummy}}} &= 1, && \forall m\in\mathcal{M},\\
\sum_{v\in\mathcal{V}} x_{m,i,v} &= 1, && \forall m\in\mathcal{M},\forall i\in\mathcal{I},\\
\mathrm{t}_v^{\min}\cdot x_{m,i,v} &\le p_{m,i,v}, && \forall m\in\mathcal{M},\forall i\in\mathcal{I},\forall v\in\mathcal{V},\\
\mathrm{t}_v^{\max}\cdot x_{m,i,v} &\ge p_{m,i,v}, && \forall m\in\mathcal{M},\forall i\in\mathcal{I},\forall v\in\mathcal{V},\\
s_{m,i_{\max}} + \sum_{v\in\mathcal{V}} p_{m,i_{\max},v} &= \mathrm{h}, && \forall m\in\mathcal{M},\\
x_{m,i,v} &\le \sum_{v'\in\text{Next}(v)} x_{m,i+1,v'}, && \\
& && \forall m\in\mathcal{M}, i\in\{1,\ldots,i_{\max}-1\},\\
x_{m,i,v'} &\le \sum_{v\in\text{Prev}(v')} x_{m,i-1,v}, && \\
& && \forall m\in\mathcal{M}, i\in\{2,\ldots,i_{\max}\},\\
x_{m,i,v} + x_{m,i+1,v'} - 1 &\le z_{m,i,v,v'}, && \forall m\in\mathcal{M},\forall i\in\{1,\ldots,i_{\max}-1\},\forall v,v'\in\mathcal{V},\\
\sum_{v\in\mathcal{V}}\sum_{v'\in\mathcal{V}} z_{m,i,v,v'} &\le 1, && \forall m\in\mathcal{M},\forall i\in\{1,\ldots,i_{\max}-1\},\\
\sum_{m\in\mathcal{M}}\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} y_{j,m,i,v} &= 1, && \forall j\in\mathcal{J},\\
\mathrm{r}_j &\le s_j, && \forall j\in\mathcal{J},\\
\mathrm{d}_j &\ge s_j + \sum_{m\in\mathcal{M}}\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} (y_{j,m,i,v}\cdot \mathrm{p}_{j,v}), && \\
& && \forall j\in\mathcal{J},\\
s_{m,i} &\le s_j + \mathrm{K}_1\cdot\left(1 - \sum_{v\in\mathcal{V}} y_{j,m,i,v}\right), && \\
& && \forall m\in\mathcal{M}, i\in\mathcal{I}, j\in\mathcal{J},\\
s_j + y_{j,m,i,v}\cdot \mathrm{p}_{j,v} &\le s_{m,i} + p_{m,i,v} + \mathrm{K}_2\,(1 - y_{j,m,i,v}), && \\
& && \forall m\in\mathcal{M}, i\in\mathcal{I}, j\in\mathcal{J}, v\in\mathcal{V},\\
s_j + \sum_{v\in\mathcal{V}} y_{j,m,i,v}\cdot \mathrm{p}_{j,v} &\le s_{j'} + \mathrm{K}_3\cdot\Bigg(3 - \sum_{v\in\mathcal{V}} y_{j,m,i,v} && \\
& \quad - \sum_{v\in\mathcal{V}} y_{j',m,i,v} - a_{j,j'}\Bigg), && \\
& && \forall m\in\mathcal{M}, i\in\mathcal{I}, \forall j,j'\in\mathcal{J},\\
a_{j,j'} &= 1 - a_{j',j}, && \forall j,j'\in\mathcal{J}, j\ne j',\\
\sum_{j\in\mathcal{J}} \left(y_{j,m,i,v}\cdot \mathrm{p}_{j,v}\right) &\le p_{m,i,v}, && \forall m\in\mathcal{M}, i\in\mathcal{I}, v\in\mathcal{V},\\
\sum_{j\in\mathcal{J}}\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} \left(y_{j,m,i,v}\cdot \mathrm{p}_{j,v}\right) &\ge \sum_{j\in\mathcal{J}}\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} \left(y_{j,m+1,i,v}\cdot \mathrm{p}_{j,v}\right), && \\
& && \forall m\in\{1,\ldots,M-1\}.
\end{aligned}
\tag{3.22}
$$

Variables have following domains:

$$
\begin{aligned}
&x_{m,i,v}\in\{0,1\},\ \forall m\in\mathcal{M}, i\in\mathcal{I}, v\in\mathcal{V}, && p_{m,i,v}\in\{0,1,\ldots,\mathrm{h}\}, \forall m\in\mathcal{M}, i\in\mathcal{I}, v\in\mathcal{V},\\
&y_{j,m,i,v}\in\{0,1\},\ \forall j\in\mathcal{J}, m\in\mathcal{M}, i\in\mathcal{I}, v\in\mathcal{V}, && s_j\in\{0,1,\ldots,\mathrm{h}\},\ \forall j\in\mathcal{J}.\\
&z_{m,i,v,v'}\in\{0,1\},\ \forall m\in\mathcal{M}, i\in\{1,\ldots,i_{\max}-1\}, v\in\mathcal{V}, v'\in\mathcal{V},\\
&a_{j,j'}\in\{0,1\},\ \forall j\in\mathcal{J}, j'\in\mathcal{J},
\end{aligned}
$$

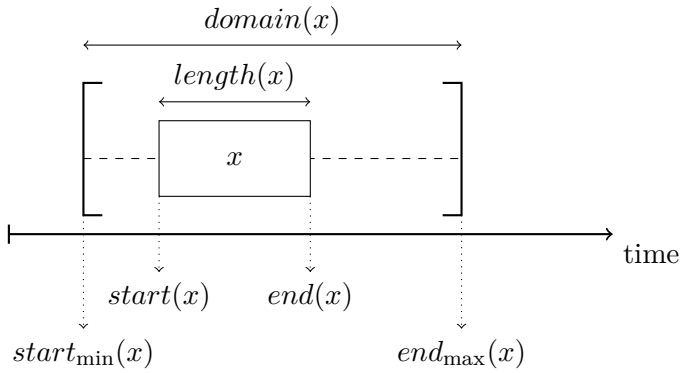


Figure 3.2: Interval variable x, which can have values from $domain(x)$, currently representing integer interval $[start(x), end(x))$ of length $length(x)$

3.1.2 CP model

In the previous section, MILP model was described; now, it will be reformulated as a CP model. Structure of this section is the same as for the MILP model; variables will be stated, followed by constraints and the objective.

Multiple different formulations of the CP model are possible, it could even look the same as the MILP model because CP supports binary and integer variables together with linear constraints. However, such a model would be quite inefficient (as I actually tested). The CP solvers (especially the ILOG CP Optimizer) are optimized to work with the interval variables as they were developed with the scheduling problems in mind. So the formulation of the CP model is different compared to the MILP model, but a lot of effort was actually made to make it efficient.

Variables

A basic building block of the CP model is an *interval variable*. It is used to model an interval of time during which a particular property holds (job is processed, a machine is idle/processing/off, ...) [46]. Its value is an integer interval $[\text{start}, \text{end})$, which is compactly represented inside the optimizer. An interval can be optional, which allows modelling of optional activities, alternative execution modes, etc. If an optional variable is not present in the solution, its length is set to 0 by the solver. Individual constraints can restrict starting/ending time of an interval, its length and relations between multiple intervals. A single interval variable is illustrated in Figure 3.2.

A composition of variables stays the same as for the MILP model, but the situation simplifies as start times and durations are implicitly represented by the interval variables. To emphasise the similarity (and not to complicate the notation), variables are again denoted by $\hat{x}$ and $\hat{y}$, subscripted by corresponding symbols for machines/intervals/modes/jobs; to distinguish them from the MILP variables, symbol $\hat{}$ was added. The meaning of the

variables is following:

$\hat{x}_{m,i,v}$ ... an optional interval representing machine m working in mode v during interval i,
$\hat{y}_{j,m,i,v}$ ... an optional interval representing job j processed by machine m in mode v during interval i.

Constraints

Starting by the description of machine schedules, each machine has to start operating in mode v_{init} and end its operation in mode v_{term} or v_{dummy}. This can be achieved by setting *presence* of corresponding interval variables to 1, making them present; see (3.23) and (3.24), respectively. Each machine must have one active mode per interval, (3.25). Time spent on interval corresponding to mode v is restricted by $\mathrm{t}_v^{\min}$, (3.26), and $\mathrm{t}_v^{\max}$, (3.27). Finally, the first interval has to start at time 0, (3.28), and the last one has to end at time h, (3.29).

$$presence(\hat{x}_{m,1,v_{\text{init}}}) = 1, \quad \forall m \in \mathcal{M}, \tag{3.23}$$

$$presence(\hat{x}_{m,i_{\max},v_{\text{dummy}}}) + presence(\hat{x}_{m,i_{\max},v_{\text{term}}}) = 1, \quad \forall m \in \mathcal{M}, \tag{3.24}$$

$$\sum_{v \in \mathcal{V}} presence(\hat{x}_{m,i,v}) = 1, \quad \forall m \in \mathcal{M}, \forall i \in \mathcal{I}, \tag{3.25}$$

$$size_{\min}(\hat{x}_{m,i,v}) = \mathrm{t}_v^{\min}, \quad \forall m \in \mathcal{M}, \forall i \in \mathcal{I}, \forall v \in \mathcal{V}, \tag{3.26}$$

$$size_{\max}(\hat{x}_{m,i,v}) = \mathrm{t}_v^{\max}, \quad \forall m \in \mathcal{M}, \forall i \in \mathcal{I}, \forall v \in \mathcal{V}, \tag{3.27}$$

$$start(\hat{x}_{m,1,v}) = 0, \quad \forall m \in \mathcal{M}, \forall v \in \mathcal{V}, \tag{3.28}$$

$$end(\hat{x}_{m,i_{\max},v}) = \mathrm{h}, \quad \forall m \in \mathcal{M}, \forall v \in \mathcal{V}. \tag{3.29}$$

It is easy to see that constraints (3.23)–(3.29) are very similar to (3.2)–(3.7). The difference is that decision variables x of MILP model were replaced by the $presence(\cdot)$ function of the CP solver and variables s and p changed to $start(\cdot)$ and $length(\cdot)$, respectively.

The next step is to add transitions. ILOG CP Optimizer offers functions that can be used for this purpose, one of them is $endAtStart(var_1, var_2, \Delta t)$, which forces interval variable var_2 to start exactly Δt time units after $end(var_1)$. Because the transition time $\mathrm{t}_{v,v'}$ is defined to be ∞ if there is no edge between modes v and v' (for implementation purposes, it could be $(\mathrm{h}+1)$), the transition constraints can be written simply as (3.30), (3.31) and (3.32), where $\mathrm{Next}(v) = \{v' \in \mathcal{V} \mid \exists e \in \mathcal{E} \text{ s.t. } e = (v, v')\}$ and $\mathrm{Prev}(v') = \{v \in \mathcal{V} \mid \exists e \in \mathcal{E} \text{ s.t. } e = (v, v')\}$. One of (3.30) and (3.31) is, in fact, redundant; however, it can tighten the model. Constraints (3.30) and (3.31) ensure that only the feasible transitions

are made (if there is some mode scheduled, that mode and the next/previous mode in the schedule have to be directly connected in the transition graph). Constraints (3.32) incorporate the transition times. Note that if at least one of the two modes constrained by $endAtStart(\cdot,\cdot,\cdot)$ function is not present, the constraint is trivially satisfied.

$$presence(\hat{x}_{m,i,v}) \leq \sum_{v' \in \mathrm{Next}(v)} presence(\hat{x}_{m,i+1,v'}), \quad \forall m \in \mathcal{M}, i \in \{1,\ldots,i_{\max}-1\}, \tag{3.30}$$

$$presence(\hat{x}_{m,i,v'}) \leq \sum_{v \in \mathrm{Prev}(v')} presence(\hat{x}_{m,i-1,v}), \quad \forall m \in \mathcal{M}, i \in \{2,\ldots,i_{\max}\}, \tag{3.31}$$

$$endAtStart(\hat{x}_{m,i-1,v}, \hat{x}_{m,i,v'}, \mathrm{t}_{v,v'}), \quad \forall m \in \mathcal{M}, \forall i \in \{2,\ldots,i_{\max}\}, \forall v, v' \in \mathcal{V}. \tag{3.32}$$

What remains now is the description of jobs. The $start_{\min}(\cdot)$ and $end_{\max}(\cdot)$ can be set to release time (3.33) and deadline (3.34), respectively, for each interval variable $\hat{y}$. Moreover, its length can be set to the processing time of the respective job in the associated mode (3.35). Still, each job has to be scheduled exactly once (3.36).

$$start_{\min}(\hat{y}_{j,m,i,v}) = \mathrm{r}_j, \quad \forall j \in \mathcal{J}, \forall m \in \mathcal{M}, \forall i \in \mathcal{I}, \forall v \in \mathcal{V}, \tag{3.33}$$

$$end_{\max}(\hat{y}_{j,m,i,v}) = \mathrm{d}_j, \quad \forall j \in \mathcal{J}, \forall m \in \mathcal{M}, \forall i \in \mathcal{I}, \forall v \in \mathcal{V}, \tag{3.34}$$

$$length(\hat{y}_{j,m,i,v}) = \mathrm{p}_{j,v}, \quad \forall j \in \mathcal{J}, \forall m \in \mathcal{M}, \forall i \in \mathcal{I}, \forall v \in \mathcal{V}, \tag{3.35}$$

$$\sum_{m \in \mathcal{M}} \sum_{i \in \mathcal{I}} \sum_{v \in \mathcal{V}} presence(\hat{y}_{j,m,i,v}) = 1, \quad \forall j \in \mathcal{J}. \tag{3.36}$$

Note that contrary to the MILP model, where there was only a single start time variable associated with each job and the processing time of a job changed according to the associated mode, each interval variable in the CP model has its start time, end time and length and so expression (3.34) is simpler to write than (3.14).

Restricting the jobs inside the corresponding interval is also effortless, because functions $startBeforeStart(var_1, var_2, \Delta t)$, interpreted as $start(var_1) + \Delta t \leq start(var_2)$, and $endBeforeEnd(var_1, var_2, \Delta t)$, interpreted as $end(var_1) + \Delta t \leq end(var_2)$, can be used. Hence, constraints (3.37) link start time of a job and interval and similarly (3.38) link their end times. Again, if at least one of the interval variables was not present, the constraint would be satisfied trivially – in consequence, one other constraint has to be added to ensure that if interval variable $\hat{y}_{j,m,i,v}$ is present, then the corresponding $\hat{x}_{m,i,v}$ will be present too. The constraint is written as implication (3.39); this form is natively supported by ILOG CP Optimizer.

$$startBeforeStart(\hat{x}_{m,i,v}, \hat{y}_{j,m,i,v}, 0) \quad \forall j \in \mathcal{J}, m \in \mathcal{M}, i \in \mathcal{I}, v \in \mathcal{V}, \tag{3.37}$$

$$endBeforeEnd(\hat{y}_{j,m,i,v}, \hat{x}_{m,i,v}, 0) \quad \forall j \in \mathcal{J}, m \in \mathcal{M}, i \in \mathcal{I}, v \in \mathcal{V}, \tag{3.38}$$

$$ifThen(presence(\hat{y}_{j,m,i,v}), presence(\hat{x}_{m,i,v})) \quad \forall j \in \mathcal{J}, m \in \mathcal{M}, i \in \mathcal{I}, v \in \mathcal{V}. \tag{3.39}$$

Finally, the last set of constraints has to forbid the overlaps of individual jobs. This was probably the most complicated part of the MILP model, but inside of a CP model it can be expressed easily, using $noOverlap(\cdot)$ function (3.40), which is applied to a set of interval variables forbidding them from overlapping. It could be applied only to individual intervals too, but in order to minimize the number of those constraints, it was applied to the whole machines.

$$noOverlap(\{\hat{y}_{j,m,i,v} \mid j \in \mathcal{J}, i \in \mathcal{I}, v \in \mathcal{V}\}), \quad \forall m \in \mathcal{M}. \tag{3.40}$$

Objective

An objective of the CP model can be written as (3.41). Note that contrary to the MILP model, the objective is not linear as $presence(\cdot)$ is not set for the individual variables. Anyway, it is not such a problem because CP optimizer can handle non-linear objectives.

$$\begin{aligned} &\sum_{m \in \mathcal{M}} \sum_{i \in \mathcal{I}} \sum_{v \in \mathcal{V}} length(\hat{x}_{m,i,v}) \cdot \mathrm{w}_v \\ &+ \sum_{m \in \mathcal{M}} \sum_{i \in \{1,\ldots,i_{\max}-1\}} \sum_{v \in \mathcal{V}} \sum_{v' \in \mathcal{V}} presence(\hat{x}_{m,i,v}) \cdot presence(\hat{x}_{m,i+1,v'}) \cdot \mathrm{c}_{v,v'} \end{aligned} \tag{3.41}$$

Additional constraints

Similarly to the MILP model, additional constraints linking jobs and machine modes (3.42) and symmetry breaking constraints (3.43) are formulated.

$$\sum_{j \in \mathcal{J}} length(\hat{y}_{j,m,i,v}) \leq length(\hat{x}_{m,i,v}), \quad \forall m \in \mathcal{M}, \forall i \in \mathcal{I}, \forall v \in \mathcal{V}, \tag{3.42}$$

$$\sum_{j \in \mathcal{J}} \sum_{i \in \mathcal{I}} \sum_{v \in \mathcal{V}} length(\hat{y}_{j,m,i,v}) \geq \sum_{j \in \mathcal{J}} \sum_{i \in \mathcal{I}} \sum_{v \in \mathcal{V}} length(\hat{y}_{j,m+1,i,v}), \quad \forall m \in \{1, \ldots, M-1\}. \tag{3.43}$$

Complete model

Now, all of the constraints will be joined with the objective to form a single CP model. Since ILOG CP Optimizer is designed to model scheduling problems, the model is much more compact compared to the MILP model. It is also easier to understand as there are no special "tricks" such as ordering variables $a_{j,j'}$ or *big-M* constraints.

$$
\begin{aligned}
\text{minimize } & \sum_{m\in\mathcal{M}}\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} \mathit{length}(\hat{x}_{m,i,v})\cdot \mathrm{w}_v \\
& + \sum_{m\in\mathcal{M}}\sum_{i\in\{1,\ldots,i_{\max}-1\}}\sum_{v\in\mathcal{V}}\sum_{v'\in\mathcal{V}} \mathit{presence}(\hat{x}_{m,i,v})\cdot \mathit{presence}(\hat{x}_{m,i+1,v'})\cdot \mathrm{c}_{v,v'} \\
\text{subject to:} & \\
\mathit{presence}(\hat{x}_{m,1,v_{\text{init}}}) &= 1, && \forall m\in\mathcal{M}, \\
\mathit{presence}(\hat{x}_{m,i_{\max},v_{\text{dummy}}}) & \\
+\mathit{presence}(\hat{x}_{m,i_{\max},v_{\text{term}}}) &= 1, && \forall m\in\mathcal{M}, \\
\sum_{v\in\mathcal{V}} \mathit{presence}(\hat{x}_{m,i,v}) &= 1, && \forall m\in\mathcal{M},\forall i\in\mathcal{I}, \\
\mathit{size}_{\min}(\hat{x}_{m,i,v}) &= \mathrm{t}_v^{\min}, && \forall m\in\mathcal{M},\forall i\in\mathcal{I},\forall v\in\mathcal{V}, \\
\mathit{size}_{\max}(\hat{x}_{m,i,v}) &= \mathrm{t}_v^{\max}, && \forall m\in\mathcal{M},\forall i\in\mathcal{I},\forall v\in\mathcal{V}, \\
\mathit{start}(\hat{x}_{m,1,v}) &= 0, && \forall m\in\mathcal{M},\forall v\in\mathcal{V}, \\
\mathit{end}(\hat{x}_{m,i_{\max},v}) &= \mathrm{h}, && \forall m\in\mathcal{M},\forall v\in\mathcal{V}, \\
\mathit{presence}(\hat{x}_{m,i,v}) &\le \sum_{v'\in\mathrm{Next}(v)} \mathit{presence}(\hat{x}_{m,i+1,v'}), && \\
& && \forall m\in\mathcal{M}, i\in\{1,\ldots,i_{\max}-1\}, \\
\mathit{presence}(\hat{x}_{m,i,v'}) &\le \sum_{v\in\mathrm{Prev}(v')} \mathit{presence}(\hat{x}_{m,i-1,v}), && \\
& && \forall m\in\mathcal{M}, i\in\{2,\ldots,i_{\max}\}, \\
\mathit{endAtStart}(\hat{x}_{m,i-1,v},\hat{x}_{m,i,v'},\mathrm{t}_{v,v'}), & && \forall m\in\mathcal{M},\forall i\in\{2,\ldots,i_{\max}\},\forall v,v'\in\mathcal{V}, \\
\mathit{start}_{\min}(\hat{y}_{j,m,i,v}) &= \mathrm{r}_j, && \forall j\in\mathcal{J},\forall m\in\mathcal{M},\forall i\in\mathcal{I},\forall v\in\mathcal{V}, \\
\mathit{end}_{\max}(\hat{y}_{j,m,i,v}) &= \mathrm{d}_j, && \forall j\in\mathcal{J},\forall m\in\mathcal{M},\forall i\in\mathcal{I},\forall v\in\mathcal{V}, \\
\mathit{length}(\hat{y}_{j,m,i,v}) &= \mathrm{p}_{j,v}, && \forall j\in\mathcal{J},\forall m\in\mathcal{M},\forall i\in\mathcal{I},\forall v\in\mathcal{V}, \\
\sum_{m\in\mathcal{M}}\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} \mathit{presence}(\hat{y}_{j,m,i,v}) &= 1, && \forall j\in\mathcal{J}, \\
\mathit{startBeforeStart}(\hat{x}_{m,i,v},\hat{y}_{j,m,i,v},0) & && \forall j\in\mathcal{J},m\in\mathcal{M},i\in\mathcal{I},v\in\mathcal{V}, \\
\mathit{endBeforeEnd}(\hat{y}_{j,m,i,v},\hat{x}_{m,i,v},0) & && \forall j\in\mathcal{J},m\in\mathcal{M},i\in\mathcal{I},v\in\mathcal{V}, \\
\mathit{ifThen}(\mathit{presence}(\hat{y}_{j,m,i,v}),\mathit{presence}(\hat{x}_{m,i,v})) & && \forall j\in\mathcal{J},m\in\mathcal{M},i\in\mathcal{I},v\in\mathcal{V}, \\
\mathit{noOverlap}(\{\hat{y}_{j,m,i,v}\mid j\in\mathcal{J},i\in\mathcal{I},v\in\mathcal{V}\}), & && \forall m\in\mathcal{M}, \\
\sum_{j\in\mathcal{J}} \mathit{length}(\hat{y}_{j,m,i,v}) &\le \mathit{length}(\hat{x}_{m,i,v}), && \\
& && \forall m\in\mathcal{M},\forall i\in\mathcal{I},\forall v\in\mathcal{V}, \\
\sum_{j\in\mathcal{J}}\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} \mathit{length}(\hat{y}_{j,m,i,v}) &\ge \sum_{j\in\mathcal{J}}\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} \mathit{length}(\hat{y}_{j,m+1,i,v}), && \\
& && \forall m\in\{1,\ldots,M-1\}.
\end{aligned}
\tag{3.44}
$$

All of the variables represent optional integer intervals. Variables $\hat{y}$ are constrained by the release time and the deadline of the associated job, while variables $\hat{x}$ are constrained only by the scheduling horizon: $\mathit{start}_{\min}(\hat{x}_{m,i,v}) = 0, \forall m\in\mathcal{M}, i\in\mathcal{I}, v\in\mathcal{V}$, $\mathit{end}_{\max}(\hat{x}_{m,i,v}) = \mathrm{h}, \forall m\in\mathcal{M}, i\in\mathcal{I}, v\in\mathcal{V}$.

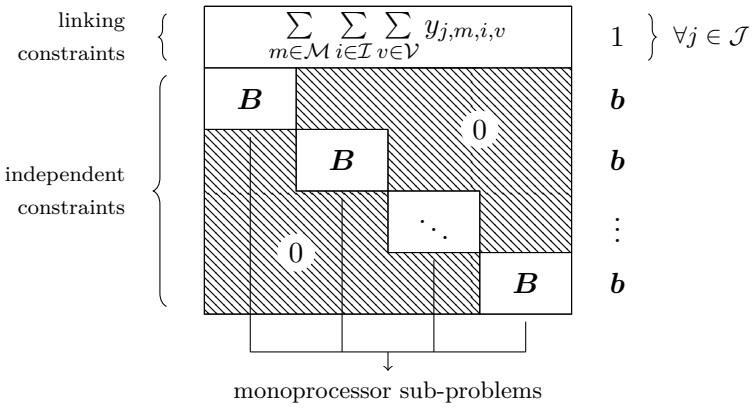


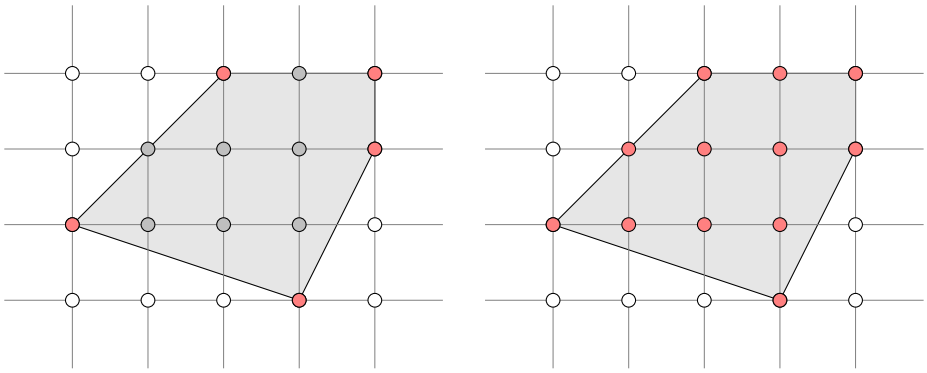
Figure 3.3: A structure of the original model before the decomposition

3.2 Branch-and-price models

In order to use a branch-and-price procedure, the original MILP model (3.22) needs to be decomposed to a master model and a pricing model. Most of the constraints of the original model hold for all machines $m \in \mathcal{M}$ individually, describing a behaviour of each of them. Even the restrictions on release times and deadlines of the jobs, which are defined globally, could be understood as individual constraints for each machine separately; if variables $s_{j,m,i,v}$ were used instead of s_j together with indicator variables forcing only one of them to be active (≥ 0) and linking them with $y_{j,m,i,v}$, then s_j would be equal to the sum of $s_{j,m,i,v}$ over all m, i, v. Similarly for $a_{j,j'}$; even though they are defined globally, only those precedences which are associated with the jobs assigned to the same machine and interval are important.

Hence, it can be seen that the only interesting connecting constraints of the original model are (3.12), forcing each job to be executed exactly once. Assuming the model where transformations $s_j \to s_{j,m,i,v}$ and $a_{j,j'} \to a_{j,j',m,i}$ were done (in order to integrate s_j and $a_{j,j'}$ into the sub-problems), decomposition can be done. Master model will ensure that each job is scheduled, whereas the decomposed sub-problems will represent single-resource scheduling problems.

Note that the original model has integer variables (linear relaxation will be used during the column generation). Its structure is illustrated in Figure 3.3. It is a special case of a more general structure described in Section 2.3.2, because the constraints (matrices) of all the sub-problems are identical. If the decomposition was used exactly as stated in Section 2.3.2, all of the polyhedrons associated with the sub-problems would be the same – in order to force integrality of the points chosen in each polyhedron (as variables of the original problem are integral) multiple variables of the same extreme point would be needed (one for each polyhedron), which might not be efficient. Also note, that requiring the integrality of the coefficients α does not need to guarantee the optimal solution of the original integer problem, because its optimal solution may be inside the convex hull (not on the boundary).



(a) (b)

Figure 3.4: Representative points (in red) for two decomposition approaches: (a) convexification, (b) discretization

An alternative approach, developed by Vanderbeck [47], can be used to solve the problems arising due to integrality. Instead of relying on the Minkowski-Weyl representation of polyhedron [32] (leading to decomposition by *convexification* described in Section 2.3.2), individual integer points can be represented alternatively [48]; that leads to so-called decomposition by *discretization*. An illustration for a bounded polyhedron is provided in Figure 3.4. The resulting model will be nearly the same as for the decomposition by convexification (of course with different representative points, as illustrated above), see [49], but convex combination coefficients α will be integral – choosing exactly one point at a time. Of course, representing the integer points directly instead of using extremal points means a potential increase of the number of variables used in the master problem; but again, the restricted master problem will operate only over their subset.

3.2.1 Master model

An aggregated decomposed master model can be formulated as (3.45)–(3.48). It is, in fact, a simple set-covering model; variables α_k are selecting individual *patterns* p_k from a set of all possible patterns $\mathcal{R}^\star$. Each pattern p_k can be understood as a column of zeros and ones with J elements $a_{j,k}$ indicating whether job j is covered by pattern k or not. Objective (3.45) minimizes the sum of costs c_k of patterns which are selected. Constraint (3.46) ensures that each job is selected exactly once. Constraint (3.47) represents aggregated convexity constraints.

$$\underset{\boldsymbol{\alpha}}{\text{minimize:}} \quad \sum_{\mathrm{p}_k \in \mathcal{R}^\star} \mathrm{c}_k \cdot \alpha_k \tag{3.45}$$

$$\text{subject to:} \quad \sum_{\mathrm{p}_k \in \mathcal{R}^\star} a_{j,k} \cdot \alpha_k = 1, \quad \forall j \in \mathcal{J}, \tag{3.46}$$

$$\sum_{\mathrm{p}_k \in \mathcal{R}^\star} \alpha_k = J, \tag{3.47}$$

$$\alpha_k \in \{0, 1\}, \qquad \forall \mathrm{p}_k \in \mathcal{R}^\star. \tag{3.48}$$

Cost c_k of pattern p_k corresponds to an optimal cost of a single-resource schedule in which exactly these jobs covered by p_k are scheduled.

Note that there could be an inequality ($\leq$) instead of an equality in constraint (3.47) because some machines might not be used; if the equality is used, there must be several empty patterns ($a_{j,\cdot} = 0 \ \forall j \in \mathcal{J}$) with a zero cost contained in $\mathcal{R}^\star$. Also note that restrictions $\alpha_k \in \{0, 1\}$ could be relaxed to $\alpha_k \in \mathbb{N}$ as selecting one pattern multiple times would not be optimal (if patterns with zero cost were not used).

Model (3.45)–(3.48) is practically identical to a master model for the vehicle routing problem with time windows described in [19]. Although the problems are clearly different, nature of linking constraints is the same. Whereas here, patterns represent set of jobs assigned to a single machine, for a vehicle routing problem, patterns represent routes taken by individual vehicles. The difference would show in the pricing problem, which is more specific (single-resource scheduling versus vehicle route finding).

Of course, using a set of all patterns $\mathcal{R}^\star$ is not feasible due to the fact that there are $\mathcal{O}(2^J)$ different patterns in total; scheduling 25 jobs implies approximately $33 \cdot 10^6$ patterns, for 30 jobs it is already 10^9, etc. Therefore, subset $\mathcal{R} \subseteq \mathcal{R}^\star$ is used, transforming the master problem to the restricted master problem, which can be written as (3.49)–(3.52).

$$\underset{\boldsymbol{\alpha}}{\text{minimize:}} \quad \sum_{\mathrm{p}_k \in \mathcal{R}} \mathrm{c}_k \cdot \alpha_k \tag{3.49}$$

$$\text{subject to:} \quad \sum_{\mathrm{p}_k \in \mathcal{R}} a_{j,k} \cdot \alpha_k \geq 1, \quad \forall j \in \mathcal{J}, \tag{3.50}$$

$$\sum_{\mathrm{p}_k \in \mathcal{R}} \alpha_k \leq J, \tag{3.51}$$

$$\alpha_k \in \mathbb{N}, \qquad \forall \mathrm{p}_k \in \mathcal{R}. \tag{3.52}$$

Linear relaxation of the RMP will be used inside the column generation procedure, but one final problem needs to be solved. At the beginning of the procedure, there are no patterns inside $\mathcal{R}$ (if some heuristic was not used to generate them); similarly, after branching occurs, some of the patterns violating new constraints imposed by the branching rule might have been deleted. In consequence, the RMP might be infeasible. To solve this

problem, additional variables will be added to the RMP in order to relax its constraints if necessary. Linear relaxation of the RMP with additional variables is formulated as (3.53)–(3.57). Two variables γ_1 and γ_2 are added, relaxing the corresponding constraint(s). Criterion is changed to integrate the new variables; K is chosen as some large number (possibly $M \cdot \mathrm{h} \cdot \max_v \mathrm{w}_v + c,\ c > 0$). If γ_1 or γ_2 is active (> 0) in the optimal solution of the RMP at the end of the column generation phase, the branch is closed as infeasible. The proposed formulation is not the only one possible, just one variable could be used to relax all of the constraints simultaneously or all of the constraints could have different variables; different formulations imply different dual constraints, therefore, influencing the optimal dual solutions (pricing vectors), see [19].

$$\underset{\boldsymbol{\alpha},\gamma_1,\gamma_2}{\text{minimize:}} \quad \sum_{\mathrm{p}_k \in \mathcal{R}} \mathrm{c}_k \cdot \alpha_k + \mathrm{K} \cdot (\gamma_1 + \gamma_2) \tag{3.53}$$

$$\text{subject to:} \quad \sum_{\mathrm{p}_k \in \mathcal{R}} a_{j,k} \cdot \alpha_k + \gamma_1 \geq 1, \qquad \forall j \in \mathcal{J}, \tag{3.54}$$

$$\sum_{\mathrm{p}_k \in \mathcal{R}} \alpha_k - \gamma_2 \leq J, \tag{3.55}$$

$$\alpha_k \geq 0, \qquad \forall \mathrm{p}_k \in \mathcal{R}, \tag{3.56}$$

$$\gamma_1 \geq 0, \gamma_2 \geq 0 \tag{3.57}$$

3.2.2 Pricing model

Now that the master model was formulated, only the formulation of the pricing problem remains. Because all of the sub-problems are identical, the pricing problem will be solved only once per each iteration. The constraints forcing the jobs to be scheduled were integrated to the master model, the remaining constraints of the MILP model (3.22) will be integrated to the pricing model; some of them will simplify. Both models (MILP and CP) will be stated here, and the differences to the original model will be discussed.

The pricing problem will be a single-resource scheduling problem (optimizing reduced cost) where the individual jobs might or might not be scheduled; its optimization criterion will consist of two parts – one reflecting the energy consumption, and the other one integrating preferences. The preferences of the jobs will be given by the RMP in each iteration. When the pricing problem is solved with the optimal objective being negative, a column will be added to the RMP. The jobs, which were scheduled in the optimal solution of the pricing problem will form the pattern of the master model; its cost will be equal to the part of the objective function which reflects the energy consumption.

MILP model

The original problem with parallel machines was decomposed – only a single machine is modelled in the sub-problem; thanks to that, variables are simplified in the following way: $x_{m,i,v} \to x_{i,v}$, $y_{j,m,i,v} \to y_{j,i,v}$, $z_{m,i,v,v'} \to z_{i,v,v'}$. The alias $s_{m,i}$ changes to s_i; variables s_j and $a_{j,j'}$ stay the same. Interpretation of the individual variables is the same as for the global model (except for the fact that there is only a single machine).

The pricing model can be written as (3.58)–(3.79). The main difference to the original model lies in the criterion (3.58), which now consists of two parts – the first one corresponds to the cost of the pattern which will be added (it is a cost of the single-resource schedule). The second part reflects preferences given by a vector of dual prices $\boldsymbol{\pi} = (\pi_0, \pi_1, \ldots, \pi_J)$, where π_0 corresponds to (3.55), and π_j corresponds to (3.54) $\forall j \in \{1, \ldots, J\}$.

Two other new types of constraints, (3.78) and (3.79), appeared in the formulation because of branching. If a branching strategy was selecting a pair of jobs (j, j') forcing them to be scheduled on the same machine in one branch and on different machines in the other one, then the pricing model of an internal node of the branching tree would have to integrate the decisions taken above (on the path from the node to the root). Defining $\mathcal{B}_{\text{same}}$ and $\mathcal{B}_{\text{diff}}$ as sets of pairs of jobs which have to be scheduled on the same machine and on different machines, respectively, constraints (3.78) and (3.79) force the model to satisfy all of the branching decisions recorded by those sets. For the root node, both sets are empty and the respective constraints are trivially satisfied.

CP model

The process of the CP pricing model creation is analogous to the MILP model creation. Most of the constraints simplify by removing the part indexing the machines ($\forall m \in \mathcal{M}$) as the pricing problem schedules only over a single machine. The objective is changed to include the preferences given by dual vector $\boldsymbol{\pi}$ of the master model. Constraints (3.101) and (3.102) are added to integrate the branching decisions into the model. The whole pricing model is written as (3.80)–(3.102).

$$\text{minimize} \quad \underbrace{\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} p_{i,v}\cdot \mathrm{w}_v + \sum_{i\in\{1,\ldots,i_{\max}-1\}}\sum_{v\in\mathcal{V}}\sum_{v'\in\mathcal{V}} z_{i,v,v'}\cdot \mathrm{c}_{v,v'}}_{\text{cost of the schedule}} \quad \underbrace{-\pi_0 - \sum_{j\in\mathcal{J}}\pi_j \sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} y_{j,i,v}}_{\text{pricing cost}} \tag{3.58}$$

subject to

$$x_{1,v_{\text{init}}} = 1, \tag{3.59}$$

$$x_{i_{\max},v_{\text{term}}} + x_{i_{\max},v_{\text{dummy}}} = 1, \tag{3.60}$$

$$\sum_{v\in\mathcal{V}} x_{i,v} = 1, \qquad \forall i\in\mathcal{I}, \tag{3.61}$$

$$\mathrm{t}_v^{\min}\cdot x_{i,v} \le p_{i,v}, \qquad \forall i\in\mathcal{I}, \forall v\in\mathcal{V}, \tag{3.62}$$

$$\mathrm{t}_v^{\max}\cdot x_{i,v} \ge p_{i,v}, \qquad \forall i\in\mathcal{I}, \forall v\in\mathcal{V}, \tag{3.63}$$

$$s_{i_{\max}} + \sum_{v\in\mathcal{V}} p_{i_{\max},v} = \mathrm{h}, \tag{3.64}$$

$$x_{i,v} \le \sum_{v'\in\text{Next}(v)} x_{i+1,v'}, \qquad \forall i\in\{1,\ldots,i_{\max}-1\}, \tag{3.65}$$

$$x_{i,v'} \le \sum_{v\in\text{Prev}(v')} x_{i-1,v}, \qquad \forall i\in\{2,\ldots,i_{\max}\}, \tag{3.66}$$

$$x_{i,v} + x_{i+1,v'} - 1 \le z_{i,v,v'}, \qquad \forall i\in\{1,\ldots,i_{\max}-1\}, \forall v,v'\in\mathcal{V}, \tag{3.67}$$

$$\sum_{v\in\mathcal{V}}\sum_{v'\in\mathcal{V}} z_{i,v,v'} \le 1, \qquad \forall i\in\{1,\ldots,i_{\max}-1\}, \tag{3.68}$$

$$\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} y_{j,i,v} \le 1, \qquad \forall j\in\mathcal{J}, \tag{3.69}$$

$$\sum_{j\in\mathcal{J}}\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} y_{j,i,v} \ge 1, \tag{3.70}$$

$$s_j \ge \mathrm{r}_j, \qquad \forall j\in\mathcal{J}, \tag{3.71}$$

$$s_j + \sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}}(y_{j,i,v}\cdot \mathrm{p}_{j,v}) \le \mathrm{d}_j, \qquad \forall j\in\mathcal{J}, \tag{3.72}$$

$$s_i \le s_j + \mathrm{K}_1\cdot\left(1 - \sum_{v\in\mathcal{V}} y_{j,i,v}\right), \qquad \forall i\in\mathcal{I}, \forall j\in\mathcal{J}, \tag{3.73}$$

$$s_j + y_{j,i,v}\cdot \mathrm{p}_{j,v} \le s_i + p_{i,v} + \mathrm{K}_2\left(1 - y_{j,m,i,v}\right), \qquad \forall j\in\mathcal{J}, \forall i\in\mathcal{I}, \forall v\in\mathcal{V}, \tag{3.74}$$

$$s_j + \sum_{v\in\mathcal{V}} y_{j,i,v}\cdot \mathrm{p}_{j,v} \le s_{j'} + \mathrm{K}_3\cdot\left(3 - \sum_{v\in\mathcal{V}} y_{j,i,v} - \sum_{v\in\mathcal{V}} y_{j',i,v} - a_{j,j'}\right), \qquad \forall i\in\mathcal{I}, \forall j,j'\in\mathcal{J}, \tag{3.75}$$

$$a_{j,j'} = 1 - a_{j',j}, \qquad \forall j,j'\in\mathcal{J}, j\ne j', \tag{3.76}$$

$$\sum_{j\in\mathcal{J}}\left(y_{j,i,v}\cdot \mathrm{p}_{j,v}\right) \le p_{i,v}, \qquad \forall i\in\mathcal{I}, \forall v\in\mathcal{V}, \tag{3.77}$$

$$\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} y_{j,i,v} = \sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} y_{j',i,v}, \qquad \forall (j,j')\in\mathcal{B}_{\text{same}}, \tag{3.78}$$

$$\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}}(y_{j,i,v} + y_{j',i,v}) \le 1, \qquad \forall (j,j')\in\mathcal{B}_{\text{diff}}. \tag{3.79}$$

$$\begin{aligned}\text{minimize}\quad & \sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}} \mathit{length}(\hat{x}_{i,v})\cdot \mathrm{w}_v \\ & + \sum_{i\in\{1,\ldots,i_{\max}-1\}}\sum_{v\in\mathcal{V}}\sum_{v'\in\mathcal{V}} \mathit{presence}(\hat{x}_{i,v})\cdot\mathit{presence}(\hat{x}_{i+1,v'})\cdot \mathrm{c}_{v,v'} \\ & -\pi_0 - \sum_{j\in\mathcal{J}}\pi_j\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}}\mathit{presence}(\hat{y}_{j,i,v})\end{aligned} \tag{3.80}$$

subject to

$$\mathit{presence}(\hat{x}_{1,v_{\text{init}}}) = 1, \tag{3.81}$$

$$\mathit{presence}(\hat{x}_{i_{\max},v_{\text{dummy}}}) + \mathit{presence}(\hat{x}_{i_{\max},v_{\text{term}}}) = 1, \tag{3.82}$$

$$\sum_{v\in\mathcal{V}}\mathit{presence}(\hat{x}_{i,v}) = 1, \qquad \forall i\in\mathcal{I}, \tag{3.83}$$

$$\mathit{size}_{\min}(\hat{x}_{i,v}) = \mathrm{t}_v^{\min}, \qquad \forall i\in\mathcal{I},\forall v\in\mathcal{V}, \tag{3.84}$$

$$\mathit{size}_{\max}(\hat{x}_{i,v}) = \mathrm{t}_v^{\max}, \qquad \forall i\in\mathcal{I},\forall v\in\mathcal{V}, \tag{3.85}$$

$$\mathit{start}(\hat{x}_{1,v}) = 0, \qquad \forall v\in\mathcal{V}, \tag{3.86}$$

$$\mathit{end}(\hat{x}_{i_{\max},v}) = \mathrm{h}, \qquad \forall v\in\mathcal{V}, \tag{3.87}$$

$$\sum_{v'\in\text{Next}(v)}\mathit{presence}(\hat{x}_{i+1,v'}) \geq \mathit{presence}(\hat{x}_{i,v}) \qquad \forall i\in\{1,\ldots,i_{\max}-1\}, \tag{3.88}$$

$$\sum_{v\in\text{Prev}(v')}\mathit{presence}(\hat{x}_{m,i-1,v}) \geq \mathit{presence}(\hat{x}_{i,v'}) \qquad \forall i\in\{2,\ldots,i_{\max}\}, \tag{3.89}$$

$$\mathit{endAtStart}(\hat{x}_{i-1,v},\hat{x}_{i,v'},\mathrm{t}_{v,v'}), \qquad \forall i\in\{2,\ldots,i_{\max}\},\forall v,v'\in\mathcal{V}, \tag{3.90}$$

$$\mathit{start}_{\min}(\hat{y}_{j,i,v}) = \mathrm{r}_j, \qquad \forall j\in\mathcal{J},\forall i\in\mathcal{I},\forall v\in\mathcal{V}, \tag{3.91}$$

$$\mathit{end}_{\max}(\hat{y}_{j,i,v}) = \mathrm{d}_j, \qquad \forall j\in\mathcal{J},\forall i\in\mathcal{I},\forall v\in\mathcal{V}, \tag{3.92}$$

$$\mathit{length}(\hat{y}_{j,i,v}) = \mathrm{p}_{j,v}, \qquad \forall j\in\mathcal{J},\forall i\in\mathcal{I},\forall v\in\mathcal{V}, \tag{3.93}$$

$$\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}}\mathit{presence}(\hat{y}_{j,i,v}), \leq 1, \qquad \forall j\in\mathcal{J}, \tag{3.94}$$

$$\sum_{j\in\mathcal{J}}\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}}\mathit{presence}(\hat{y}_{j,i,v}) \geq 1, \tag{3.95}$$

$$\mathit{startBeforeStart}(\hat{x}_{i,v},\hat{y}_{j,i,v},0) \qquad \forall j\in\mathcal{J},\forall i\in\mathcal{I},\forall v\in\mathcal{V}, \tag{3.96}$$

$$\mathit{endBeforeEnd}(\hat{y}_{j,i,v},\hat{x}_{i,v},0), \qquad \forall j\in\mathcal{J},\forall i\in\mathcal{I},\forall v\in\mathcal{V}, \tag{3.97}$$

$$\mathit{ifThen}(\mathit{presence}(\hat{y}_{j,i,v}),\mathit{presence}(\hat{x}_{i,v})), \qquad \forall j\in\mathcal{J},\forall i\in\mathcal{I},\forall v\in\mathcal{V}, \tag{3.98}$$

$$\mathit{noOverlap}(\{\hat{y}_{j,i,v} \mid j\in\mathcal{J}, i\in\mathcal{I}, v\in\mathcal{V}\}), \tag{3.99}$$

$$\sum_{j\in\mathcal{J}}\mathit{length}(\hat{y}_{j,i,v}) \leq \mathit{length}(\hat{x}_{i,v}), \qquad \forall i\in\mathcal{I},\forall v\in\mathcal{V}, \tag{3.100}$$

$$\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}}\mathit{presence}(\hat{y}_{j,i,v}) = \sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}}\mathit{presence}(\hat{y}_{j',i,v}), \qquad \forall(j,j')\in\mathcal{B}_{\text{same}}, \tag{3.101}$$

$$\sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}}\mathit{presence}(\hat{y}_{j,i,v}) + \sum_{i\in\mathcal{I}}\sum_{v\in\mathcal{V}}\mathit{presence}(\hat{y}_{j',i,v}) \leq 1, \qquad \forall(j,j')\in\mathcal{B}_{\text{diff}}. \tag{3.102}$$

3.3 Reference model

Solution approaches studied in this work are designed to find an optimal solution to the problem. In order to compare them to existing works, a representative reference MILP model was chosen and adapted to fit the problem statement. Several authors addressed similar problems – explicitly modelling the states of the machines. Mitra et al. [5] schedule discrete operating modes of plant components; however, continuous production is assumed, which does not fit the settings discussed here. Selmair et al. [10] formulate a model for a job shop production system and finally Shrouf et al. [9] develop a model for single machine production scheduling. None of the models fits the problem discussed here perfectly, but all of them share the property of being based on a time-indexed formulation. Model described by Shrouf et al. was selected for comparison because it can be adapted easily, extending their single-machine formulation to a parallel-resource formulation.

Several simplifications of the problem statement are made in accordance with the original formulation described in [9]. It is assumed, that there is only one processing mode (i.e. mode in which jobs can be processed). Furthermore, it is assumed that $\mathrm{t}_v^{\min} = 0$ and $\mathrm{t}_v^{\max} = \mathrm{h}$ for all $v \in \mathcal{V}$.

On the other hand, the original model worked under the assumption that jobs are scheduled in the fixed order; that is relaxed there, allowing arbitrary order of the jobs to be scheduled. That increases the difficulty of the problem because the original formulation covered only a subset of all possible orderings.

One of the differences between the selected model and the global model proposed here is the fixed number of intervals assumed by the global model. As the original time-indexed model does not have these restrictions, its optimal objective could be better than the one found by the global model described here. However, having arbitrary number of switches is not usually technologically feasible, and the global model described here could not be build while not having the maximal number of the switches defined (which is one of its disadvantages). Therefore, for the sake of comparison between the two models, restrictions on the maximal number of switches between modes on a single machine are added to the reference model. Technically, that simplifies the problem, because it eliminates some of the possible solutions, reducing the size of the search space.

To be consistent with the formulations of the global models, variables with analogous meaning will be denoted by the same letters, adding a symbol ˜ above the variables of the reference model. As the model is time-indexed, the scheduling horizon is discretized into h periods $p \in \mathcal{P}$, $\mathcal{P} = \{1, \ldots, \mathrm{h}\}$; a machine mode or a transition is defined for each of them.

Variables

Five types of binary variables are used in the model. They are:

$$\tilde{x}_{m,p,v} = \begin{cases} 1 & \text{if machine } m \text{ operates in mode } v \text{ during period } p, \\ 0 & \text{otherwise,} \end{cases}$$

$$\tilde{z}_{m,p,v,v'} = \begin{cases} 1 & \text{if machine } m \text{ changes its mode from } v \text{ to } v' \text{ during} \\ & \text{period } p, \\ 0 & \text{otherwise,} \end{cases}$$

$$\tilde{w}_{m,p,v} = \begin{cases} 1 & \text{if mode } v \text{ starts on machine } m \text{ during period } p \\ 0 & \text{otherwise,} \end{cases}$$

$$\tilde{y}_{j,m,p} = \begin{cases} 1 & \text{if job } j \text{ is processed by machine } m \text{ during period } p, \\ 0 & \text{otherwise,} \end{cases}$$

$$\tilde{s}_{j,m,p} = \begin{cases} 1 & \text{if job } j \text{ starts to be processed by machine } m \text{ during} \\ & \text{period } p \\ 0 & \text{otherwise.} \end{cases}$$

In fact, binary variables $\tilde{w}_{m,p,v}$ have similar interpretation to $s_{m,i,v}$, but not to confuse them with $\tilde{s}_{j,m,p}$, different symbol was used.

Model

Note that for the reference model, the transition graph is not transformed (dummy vertex is not used) and is it assumed that the transition graph does not contain any reflexive edge. Denoting the only processing mode by v_{proc}, the adapted reference model can be written as (3.103)–(3.122).

The objective function (3.103) integrates the cost of modes and the cost of transitions. Constraint (3.104) requires that each machine must either be in on of the modes or in a transition between them in each period. Inequalities (3.105) and (3.106) limit mode or transition in which a machine can operate, if it was operating in a given mode/transition in the previous period. Constraints (3.107) and (3.108) establish lower and upper bound on the number of transition periods, such that the number of periods spent on the transition is exactly equal to the transition length. Constraints (3.109)–(3.111) link the mode $\tilde{x}_{m,p,v}$ and its start time $\tilde{w}_{m,p,v}$. Expression (3.112) restricts the maximal number of intervals (switches between modes). Equalities (3.113)–(3.115) set an initial mode for the first period (also setting start variable $\tilde{w}$ to 1) and a terminal mode for the last period. The remaining constraints relate to the jobs: forcing the processing mode when a job is processed (3.116), allowing processing of at most one job per period (3.117) and at most one start per job (3.118), integrating release times, deadlines and processing times (3.119)–(3.121) and forcing job to be processed without preemption (3.122).

$$\text{minimize} \sum_{m\in\mathcal{M}}\sum_{p\in\mathcal{P}}\sum_{v\in\mathcal{V}} \tilde{x}_{m,p,v}\cdot \mathrm{w}_v + \sum_{m\in\mathcal{M}}\sum_{p\in\mathcal{P}}\sum_{v\in\mathcal{V}}\sum_{v'\in\mathcal{V}} \tilde{z}_{m,p,v,v'}\cdot \mathrm{c}_{v,v'} \tag{3.103}$$

subject to

$$\sum_{v\in\mathcal{V}} \tilde{x}_{m,p,v} + \sum_{v\in\mathcal{V}}\sum_{v'\in\mathcal{V}} \tilde{z}_{m,p,v,v'} = 1, \qquad \forall m\in\mathcal{M}, \forall p\in\mathcal{P}, \tag{3.104}$$

$$\tilde{x}_{m,p,v} \leq \left(\tilde{x}_{m,p+1,v} + \sum_{\substack{v'\in\mathrm{Next}(v)\\ \mathrm{t}_{v,v'}=0}} \tilde{x}_{m,p+1,v'} + \sum_{\substack{v'\in\mathrm{Next}(v)\\ \mathrm{t}_{v,v'}>0}} \tilde{z}_{m,p+1,v,v'} \right), \qquad \forall m\in\mathcal{M}, \forall p\in\{1,\dots,\mathrm{h}-1\}, \forall v\in\mathcal{V}, \tag{3.105}$$

$$\tilde{z}_{m,p,v,v'}, \leq \tilde{x}_{m,p+1,v'} + \tilde{z}_{m,p+1,v,v'} \qquad \forall m\in\mathcal{M}, \forall p\in\{1,\dots,\mathrm{h}-1\}, \forall v,v'\in\mathcal{V}, \tag{3.106}$$

$$(\tilde{x}_{m,p,v} + \tilde{z}_{m,p+1,v,v'} - 1)\cdot \mathrm{t}_{v,v'} \leq \sum_{p'=p+1}^{\min\{p+\mathrm{t}_{v,v'},\mathrm{h}\}} \tilde{z}_{m,p',v,v'}, \qquad \forall m\in\mathcal{M}, \forall p\in\{1,\dots,\mathrm{h}-1\}, \forall v,v'\in\mathcal{V}, v'\in\mathrm{Next}(v), \mathrm{t}_{v,v'}>0, \tag{3.107}$$

$$\tilde{z}_{m,p,v,v'} + \tilde{z}_{m,p+\mathrm{t}_{v,v'},v,v'} \leq 1, \qquad \forall m\in\mathcal{M}, \forall p\in\mathcal{P}, \forall v,v'\in\mathcal{V}, 0<p+\mathrm{t}_{v,v'}\leq \mathrm{h}, \tag{3.108}$$

$$\tilde{x}_{m,p,v} + \sum_{\substack{v'\in\mathcal{V}\\ v'\neq v}} \tilde{x}_{m,p-1,v'} + \sum_{v'\in\mathcal{V}}\sum_{v''\in\mathcal{V}} \tilde{z}_{m,p-1,v',v''} \leq \tilde{w}_{m,p,v} + 1, \qquad \forall m\in\mathcal{M}, \forall p\in\{2,\dots,\mathrm{h}\}, \forall v\in\mathcal{V}, \tag{3.109}$$

$$\tilde{x}_{m,p-1,v} + \tilde{x}_{m,p,v} \leq (1-\tilde{w}_{m,p,v}) + 1, \qquad \forall m\in\mathcal{M}, \forall p\in\{2,\dots,\mathrm{h}\}, \forall v\in\mathcal{V}, \tag{3.110}$$

$$\tilde{w}_{m,p,v} \leq \tilde{x}_{m,p,v}, \qquad \forall m\in\mathcal{M}, \forall p\in\mathcal{P}, \forall v\in\mathcal{V}, \tag{3.111}$$

$$\sum_{p\in\mathcal{P}}\sum_{v\in\mathcal{V}} \tilde{w}_{m,p,v} \leq i_{\max}, \qquad \forall m\in\mathcal{M}, \tag{3.112}$$

$$\tilde{x}_{m,1,v_{\mathrm{init}}} = 1, \qquad \forall m\in\mathcal{M}, \tag{3.113}$$

$$\tilde{w}_{m,1,v_{\mathrm{init}}} = 1, \qquad \forall m\in\mathcal{M}, \tag{3.114}$$

$$\tilde{x}_{m,\mathrm{h},v_{\mathrm{term}}} = 1, \qquad \forall m\in\mathcal{M}, \tag{3.115}$$

$$\sum_{j\in\mathcal{J}} \tilde{y}_{j,m,p} \leq \tilde{x}_{m,p,v_{\mathrm{proc}}}, \qquad \forall m\in\mathcal{M}, \forall p\in\mathcal{P}, \tag{3.116}$$

$$\sum_{j\in\mathcal{J}} \tilde{y}_{j,m,p} \leq 1, \qquad \forall m\in\mathcal{M}, \forall p\in\mathcal{P}, \tag{3.117}$$

$$\sum_{m\in\mathcal{M}}\sum_{p\in\mathcal{P}} \tilde{s}_{j,m,p} = 1, \qquad \forall j\in\mathcal{J}, \tag{3.118}$$

$$\sum_{m\in\mathcal{M}}\sum_{p\in\mathcal{P}} \tilde{y}_{j,m,p} = \mathrm{p}_{j,v_{\mathrm{proc}}}, \qquad \forall j\in\mathcal{J}, \tag{3.119}$$

$$\tilde{s}_{j,m,p} = 0, \qquad \forall j\in\mathcal{J}, \forall m\in\mathcal{M}, \forall p\in\mathcal{P}, p<\mathrm{r}_j, \tag{3.120}$$

$$\tilde{s}_{j,m,p} = 0, \qquad \forall j\in\mathcal{J}, \forall m\in\mathcal{M}, \forall p\in\mathcal{P}, p>\mathrm{d}_j - \mathrm{p}_{j,v_{\mathrm{proc}}}, \tag{3.121}$$

$$\sum_{p'=p}^{\min\{p+\mathrm{p}_{j,v_{\mathrm{proc}}}-1,\mathrm{h}\}} \tilde{y}_{j,m,p'} \geq \mathrm{p}_{j,v_{\mathrm{proc}}}\cdot \tilde{s}_{j,m,p}, \qquad \forall j\in\mathcal{J}, \forall m\in\mathcal{M}, \forall p\in\mathcal{P}. \tag{3.122}$$

4 Experiments

For the comparison of the individual models/solution techniques, various experiments were performed. Some of them are described in this chapter – showing the performance of the respective approaches. A structure of the chapter is following: firstly, branch-and-price settings and a data generation procedure are described; secondly, the global MILP model is compared to the reference MILP model; and finally, comparison of the proposed approaches is given.

All experiments were done on a personal PC with Intel Core i5 having 2 physical (4 virtual) cores and 8 GB RAM. As for the solvers – Gurobi 8.0 solver was used for LP/MILP models, while IBM ILOG CP Optimizer (12.8) was used for CP models.

4.1 Branch-and-price settings

For the initialization of a branch-and-price procedure, a simple two-stage heuristic is used. The first stage is based on a CP model, which creates optional interval variables for each job (for each machine), setting their minimal starts to the release time, maximal ends to the deadline and lengths to the minimal processing time across all modes. The model is looking for an assignment of jobs to the individual machines. If none is found – the instance is infeasible (as the release times and the deadlines do not permit any assignment even for the fastest processing times). If some assignment is found, then MILP model similar to the pricing model (except for the criterion, which does not include preferences, and for the fact, that each given job has to be scheduled) is called to find an optimal monoprocessor solution to each found assignment (for each machine); that is the second stage. The found assignments are, in fact, patterns, and their optimal solutions correspond to the patterns costs. It might happen, that the heuristic does not find any solution in the second stage. For that reason, simple patterns corresponding to the individual jobs (one pattern per one job) are added to the initial set too.

The branching rule selects two different jobs which occurred in the patterns whose optimal coefficients in the master model were fractional. The pair with maximal overlap (considering their release times and deadlines), which was not selected by decisions taken on the path from the root node to the current node, is selected. Two branches are created afterwards – forcing the selected jobs to be scheduled (i) on the same machine and (ii) on different machines (while maintaining all of the previous decisions, taken on the path from the root node to the current node, too).

4.2 Data generation

Problems of energy optimization in scheduling have been studied only recently and even though multiple works exists is the field, they often concentrate on a specific problem. As there is no unifying frame, there are none benchmark instances (at least to the best of my knowledge). For the purpose of testing, random instances of the problem will be generated.

For given transition graph G (with a single processing mode) and maximal number of intervals $i_{\max}$, J jobs will be scheduled on M machines. Parameters of job j (release time, deadline and processing time) will be generated randomly. One of the possible generating schemes is the following one:

$$\mathrm{p}_j \sim U(a_\mathrm{p}, b_\mathrm{p}), \tag{4.1}$$

$$\mathrm{r}_j \sim U(a_\mathrm{r}, b_\mathrm{r}), \tag{4.2}$$

$$\mathrm{d}_j \sim U(a_\mathrm{d}, b_\mathrm{d}), \tag{4.3}$$

where expression $U(a, b)$ denotes the uniform integer distribution over the closed interval $[a, b]$. Of course the parameters $a.$ and $b.$ do not need to be independent nor constant (they can represent values generated from some probability distribution, for example). Specific settings/generation scheme will be described for each experiment individually.

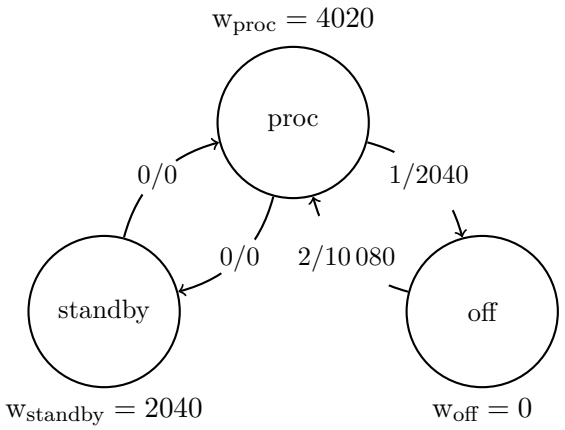


Figure 4.1: The transition graph used for Experiment 1; edges are labelled by transition time/transition cost

4.3 Experiment 1: Comparison of the global model and the reference model

This experiment aims to compare the global MILP model proposed in this work with a time-indexed MILP model proposed in [9], which was adapted to fit the problem statement discussed here (see Section 3.3).

4.3.1 Setting

A transition graph used for this experiment was originally published in [9], together with the time-indexed model, and can be seen in Figure 4.1. Costs were slightly changed to be divisible by 60 (for the purposes of scaling – see details below).

For the experiment, instances with $J \in \{5, 10, 15\}$ jobs were generated. The jobs were scheduled on $M \in \{1, 2, 4\}$ machines. Three different settings of $i_{\max}$ were tested, $i_{\max} \in \{3, 5, 7\}$. Scheduling horizon h was set to 32 time units (same as in the original work), $h = 32$. To test the efficiency of the time-indexed model, generated instances were scaled by factor $scale \in \{1, 4, 60\}$ – transition costs/times were scaled together with the horizon and parameters of the jobs. The scaling could be interpreted as a change of the granularity of discretization, e.g., $scale = 1$ could mean discretization by hours, $scale = 4$ by 15-minutes intervals and $scale = 60$ by minutes. Scaled instances have the same optimal objective value (for fixed $i_{\max}$ and M), but the scaling might influence the size of the models as well as the solution times.

For each setting of $J \in \{5, 10, 15\}$, five different instances were generated in a following way (for $scale = 1$):

$$\mathrm{p}_j \sim U(1, 5), \tag{4.4}$$

$$\mathrm{r}_j \sim U(0, \mathrm{h} - \mathrm{p}_j - 1), \tag{4.5}$$

$$\mathrm{d}_j \sim U(\mathrm{r}_j + \mathrm{p}_j, \mathrm{h} - 1), \tag{4.6}$$

for all $j \in \{1, \ldots, J\}$. Constant (-1) is in the generation formula for r_j and d_j because of the time $\mathrm{t}_{\mathrm{proc,off}}$. These instances were scaled and solved for all combinations of $i_{\max}$ and M. A time limit was set to 120 seconds (per one instance).

4.3.2 Results

Solving times (in seconds) are shown in Table 4.1 and Table 4.2, respectively. Results for $scale = 60$ are not listed, because the time-indexed model failed to give reasonable results even for $scale = 4$; for the global model, results were similar to $scale = 4$ listed in Table 4.1.

For the tested instances, the global model performs similarly for all of the discretization granularities. It is because the size of the model depends only on the number of machines/jobs/intervals/modes; the length of an interval has an impact only on the objective value – longer intervals does not imply larger model. The largest tested global models had about 1000 constraints and 500 variables.

On the other hand, the size of the time-indexed model depends on the number of periods (not intervals), hence by increasing the precision of the discretization, the size of the model will increase too. From Table 4.2, it is evident that after scaling by a relatively small factor (4), the model becomes too large to be solved in a reasonable time. The largest time-indexed models for $scale = 4$ had about 23 000 variables and 28 000 constraints; even worse – for $scale = 60$, the models had about 340 000 variables and 420 000 constraints.

Certainly, both models have their advantages and disadvantages. For example, it is easy to integrate time-dependent costs into a time-indexed model, whereas it would be difficult for the global model as start times and lengths of individual intervals are not fixed. Furthermore, the time-indexed model can perform better on small instances – it can be seen on instances 15-3 and 15-5. Even though the global model found their optimal solutions, it failed to complete the optimality proofs. Another disadvantage of the global model is that it depends on the number of intervals $i_{\max}$ – with $i_{\max}$ increasing, more intervals need to be modelled, increasing the size of the model. Contrary to that, $i_{\max}$ appears in a form of a single constraint (per machine) in a time-indexed model. If there was no requirement on the maximal number of intervals, the reference model would work as well, becoming even simpler – because variables $\tilde{w}_{m,p,v}$ could be removed. However, a time-indexed model can be efficient only for trivially small instances. If there were many periods, even an initialization of the model would become infeasible.

Global MILP model

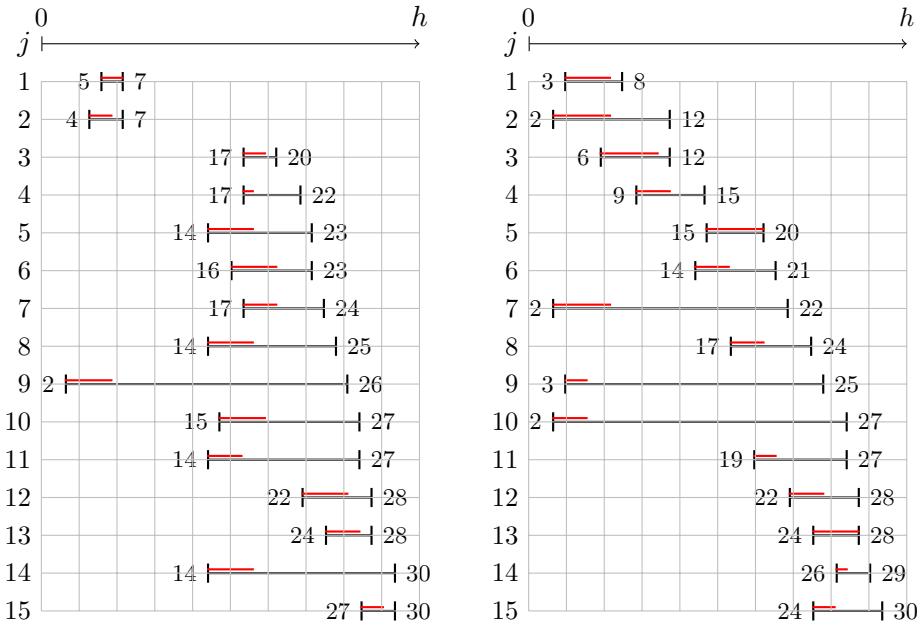
			$i_{\max}$								
			3			5			7		
			M			M			M		
		id	1	2	4	1	2	4	1	2	4
***scale* = 1**											
		1	0.02	0.02	0.00	0.03	0.08	0.06	0.06	0.09	0.09
		2	0.02	0.03	0.02	0.03	0.08	0.09	0.05	0.14	0.42
	5	3	0.00	0.02	0.03	0.02	0.06	0.18	0.03	0.14	0.91
		4	0.02	0.00	0.02	0.02	0.09	0.06	0.03	0.09	0.12
		5	0.00	0.02	0.00	0.02	0.04	0.06	0.03	0.09	0.12
		1	0.02	0.08	0.09	0.03	0.28	0.14	0.05	0.89	0.92
		2	0.00	0.02	0.03	0.00	0.14	0.16	0.02	0.59	0.22
J	10	3	0.00	0.03	0.09	0.02	0.14	0.58	0.00	0.42	0.92
		4	0.00	0.06	0.06	0.00	0.23	0.62	0.00	1.19	2.43
		5	0.00	0.03	0.09	0.03	0.13	0.95	0.03	0.44	4.76
		1	0.00	0.14	0.41	0.00	0.41	1.04	0.02	1.58	8.13
		2	0.00	0.15	0.19	0.00	0.14	0.28	0.00	0.22	0.52
	15	3	0.00	0.37	> 120	0.00	13.49	> 120	0.02	> 120	> 120
		4	0.02	0.14	0.25	0.00	0.20	0.50	0.00	1.19	0.36
		5	0.02	0.02	3.98	0.00	0.31	> 120	0.02	0.66	> 120
***scale* = 4**											
		1	0.02	0.00	0.02	0.06	0.05	0.06	0.06	0.09	0.13
		2	0.00	0.02	0.02	0.02	0.05	0.06	0.03	0.08	0.13
	5	3	0.02	0.02	0.03	0.00	0.05	0.14	0.03	0.13	0.42
		4	0.00	0.00	0.02	0.02	0.05	0.03	0.03	0.08	0.14
		5	0.00	0.02	0.02	0.00	0.03	0.03	0.03	0.11	0.09
		1	0.02	0.11	0.13	0.02	0.14	0.30	0.05	0.76	0.91
		2	0.00	0.10	0.06	0.00	0.17	0.17	0.02	0.76	0.34
J	10	3	0.00	0.02	0.05	0.00	0.16	0.59	0.00	0.39	0.92
		4	0.02	0.05	0.06	0.00	0.30	0.82	0.00	1.58	2.32
		5	0.02	0.02	0.06	0.03	0.11	0.48	0.05	0.39	3.59
		1	0.02	0.16	0.16	0.02	1.20	1.72	0.02	2.82	3.32
		2	0.00	0.03	0.20	0.00	0.17	0.13	0.02	1.05	0.62
	15	3	0.00	0.03	> 120	0.00	37.83	> 120	0.00	> 120	> 120
		4	0.00	0.16	0.28	0.02	0.17	0.50	0.00	0.53	0.25
		5	0.00	0.03	3.54	0.00	0.23	> 120	0.02	1.01	> 120

Table 4.1: Solving times (in seconds) for different settings of the maximal number of intervals $i_{\max}$, number of jobs J, number of machines M and *scale* for the global model; infeasible instances are shown in gray

Reference model

			$i_{\max}$								
			3			5			7		
			M			M			M		
		id	1	2	4	1	2	4	1	2	4
***scale* = 1**											
		1	0.04	0.35	1.09	0.08	0.17	0.50	0.05	0.20	0.52
		2	0.03	1.07	10.90	0.24	1.01	5.37	0.22	0.80	8.11
	5	3	0.00	0.02	0.41	0.00	0.03	0.44	0.00	0.02	0.44
		4	0.00	0.12	1.07	0.00	0.19	3.46	0.00	0.14	4.06
		5	0.00	0.11	0.73	0.00	0.66	0.69	0.00	0.43	0.74
		1	0.00	0.45	10.17	0.00	0.35	13.06	0.00	0.80	26.83
		2	0.02	3.60	23.00	0.03	6.08	54.28	0.02	4.26	60.03
J	10	3	0.02	0.03	0.86	0.00	0.03	1.03	0.02	0.03	0.95
		4	0.02	0.22	1.31	0.02	0.30	1.38	0.00	0.35	1.02
		5	0.00	0.02	1.46	0.00	0.02	1.58	0.00	0.02	1.82
		1	0.02	0.23	1.54	0.00	0.37	1.61	0.02	0.41	1.91
		2	0.00	0.31	3.43	0.02	1.22	7.16	0.02	0.86	4.34
	15	3	0.00	0.11	40.39	0.00	0.13	58.41	0.00	0.11	60.57
		4	0.00	0.14	2.26	0.00	0.19	2.47	0.02	0.45	2.43
		5	0.00	0.03	19.61	0.02	0.03	1.66	0.00	0.03	2.03
***scale* = 4**											
		1	1.54	109.56	> 120	4.48	82.72	> 120	13.28	76.98	> 120
		2	0.36	> 120	> 120	28.13	> 120	> 120	33.83	> 120	> 120
	5	3	0.03	0.12	> 120	0.02	0.11	> 120	0.02	0.11	> 120
		4	0.03	50.00	> 120	0.00	> 120	> 120	0.02	65.60	> 120
		5	0.02	22.66	> 120	0.02	> 120	> 120	0.00	> 120	> 120
		1	0.03	42.18	> 120	0.02	> 120	> 120	0.02	> 120	> 120
		2	0.35	> 120	> 120	0.34	> 120	> 120	0.34	> 120	> 120
J	10	3	0.02	0.14	> 120	0.02	0.13	> 120	0.02	0.14	> 120
		4	0.02	84.60	> 120	0.02	105.97	> 120	0.02	79.97	> 120
		5	0.02	0.14	> 120	0.00	0.13	> 120	0.02	0.13	> 120
		1	0.03	49.11	> 120	0.02	41.03	> 120	0.03	52.43	> 120
		2	0.02	> 120	> 120	0.02	> 120	> 120	0.02	> 120	> 120
	15	3	0.02	0.66	> 120	0.02	0.64	> 120	0.02	0.64	> 120
		4	0.02	85.07	> 120	0.03	91.43	> 120	0.02	> 120	> 120
		5	0.02	0.16	> 120	0.03	0.14	> 120	0.02	0.16	> 120

Table 4.2: Solving times (in seconds) for different settings of the maximal number of intervals $i_{\max}$, number of jobs J, number of machines M and *scale* for the reference time-indexed model; infeasible instances are shown in gray



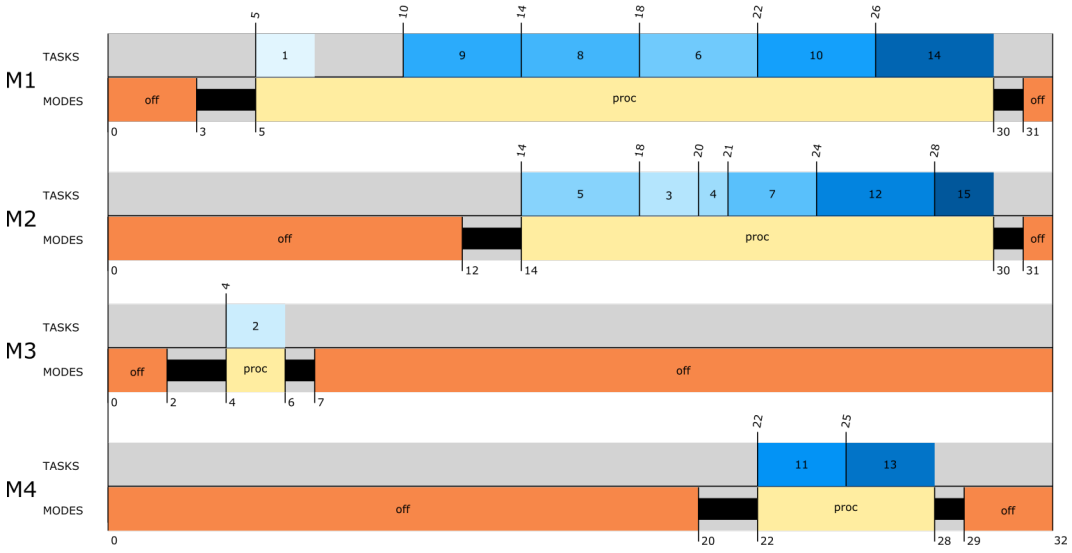
(a) 15–3 (b) 15–4

Figure 4.2: Release times and deadlines of the jobs: (a) instance 15-3, (b) instance 15-4; red lines illustrate the minimal processing times

As instance 15-3 seems to be harder to solve than the rest of the instances, its structure is shown in Figure 4.2 (a). Jobs are listed in non-decreasing order of their deadlines. It can be seen, that the difficulties probably arise because jobs 3-11 and 14 have similar time-windows implied by their release times and deadlines and thus they "compete" for the machines. Contrary to that, Figure 4.2 (b) shows the structure of instance 15-4, which was solved quickly.

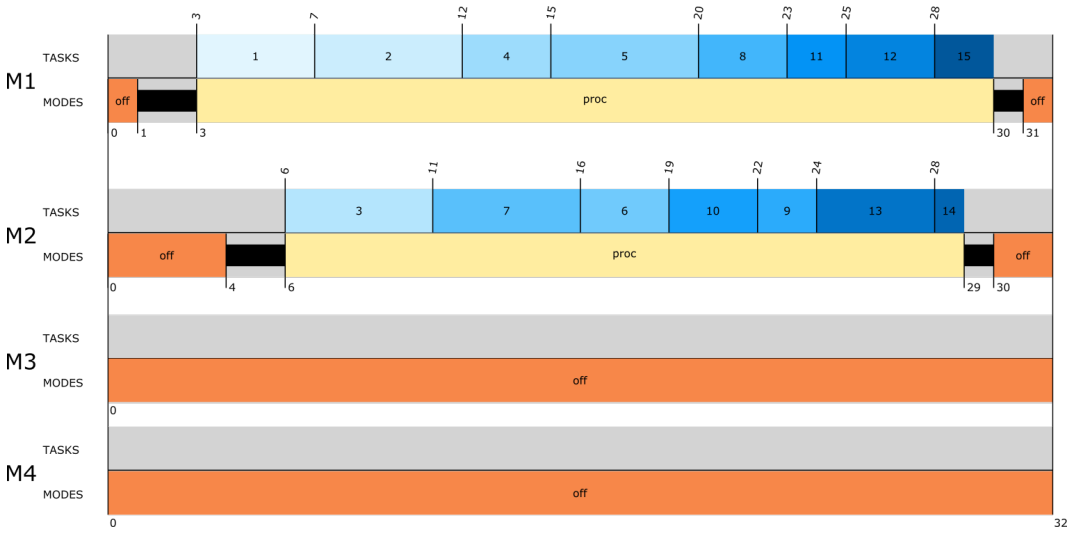
Even though both instances may look similar, the difference can be noticed after comparing their optimal solutions, see Figure 4.3. Whereas all of the machines were forced to operate for instance 15-3, it was possible to schedule tasks of instance 15-4 "nicely" one after another using only 2 machines. In this case, solver was able to find an optimal solution quickly, thus pruning many other feasible solutions. On the other hand, the structure of instance 15-3 allows many possible assignments – it is not clear, which tasks should be grouped to form the optimal solution.

Notice that there is a space between the end of task 1 and the start of task 9 in the optimal solution of 15-3. Tasks 1 and 2 cannot be processed on the same machine and so task 1 needed to be assigned to a different one. This empty space could be spent in the standby mode, however, that would require more than 3 intervals ($i_{\max} = 3$). Indeed, the optimal objective value improved from 241 380 ($i_{\max} = 3$) to 235 440 ($i_{\max} = 5$) saving additional 2 % of the costs.



(a)

15-3



(b)

15-4

Figure 4.3: Illustration of the optimal schedules for $i_{\max} = 3$: (a) instance 15-3, (b) instance 15-4

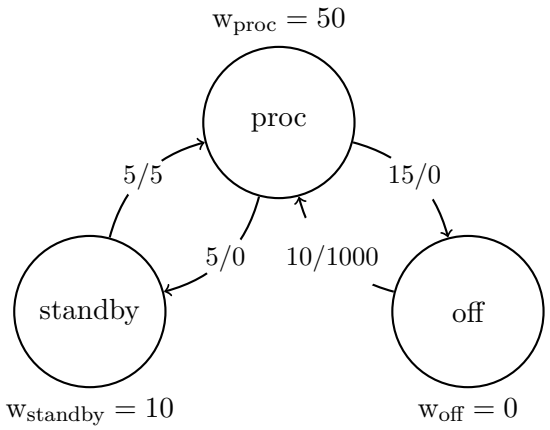


Figure 4.4: Transition graph used for the experiment 2; edges are labelled by transition times/transition costs

4.4 Experiment 2: Comparison of the proposed approaches

While working on the models and the algorithm, the individual approaches were tested on several randomly generated instances. Settings of the experiments and their results will be described in this section.

4.4.1 Settings

The transition graph used for the experiment is shown in Figure 4.4. Its structure is simple – it has one mode in which the machine is turned off (having zero power consumption), one processing mode and one standby mode. It is assumed, that $t_v^{\min}$ and $t_v^{\max}$ are 0 and h, respectively, for all $v \in \mathcal{V}$. The reason for using this topology is that in reality, machines whose internal state can be modelled in such a way really exist. For example, in SKODA Auto company, there are several identical furnaces, which can be either turned off, heated to a high temperature (processing mode used for steal hardening) or heated to a slightly lower temperature (standby mode used for energy savings) [6]. Of course the costs of the graph would be different, but the behaviour of machines would be similar as it usually takes a long time to turn a machine on (heating to a high temperature) and off (cooling to a normal temperature), while transitions between the standby mode and the processing mode are faster.

Scheduling horizon h was set to 1000 units and maximally 7 intervals were allowed, $i_{\max} = 7$. Tests were performed for $J \in \{5, 10, 15\}$ and $M \in \{1, 2, 3, 4\}$. For each combination of J and M, 5 testing instances were generated. Maximal solving time was set to 300 seconds per instance. It is assumed, that the machines need to start and end in the "off" mode.

Parameters of the jobs were obtained by iteratively creating a feasible solution – assigning jobs to machines. Processing times were generated uniformly from $U(1, 100)$ and the jobs

were iteratively added to the partial feasible schedule (random unoccupied part of the scheduling horizon was selected and the generated job/processing time was assigned to it; it was possibly trimmed, if it did not fit the selected space). Afterwards, a release time and a deadline of the job were established by growing the previously generated interval (corresponding to the processing time) by U(0,25) units to the left and U(0,25) to the right. Generated tasks for $J = 5$ and $J = 10$ are shown in Appendix A.

4.4.2 Results

The average solving times over the generated instances for each combination of M and J are shown in Table 4.3. For tested instances, both global models perform relatively well when number of machines M is small (1 or 2). However, their performance becomes worse if M is larger (3 or 4). The global CP model did not even manage to solve any instance in a given time for $J = 15$, $M \in \{3, 4\}$.

Considering results of the branch-and-price approaches, their solving times are comparable for fixed J, when M is changing. The constraints on the number of machines are integrated to the master model, while the sub-problem solves only monoprocessor problems, thus removing potential symmetries, which have an impact on the performance of the global models. Thanks to this decomposition, branch-and-price algorithm with MILP sub-problems outperformed the global models on the given dataset for $M \in \{3, 4\}$ and

M-*J*	BP MILP		BP CP		Global MILP		Global CP	
	avg	#solved	*avg*	#solved	*avg*	#solved	*avg*	#solved
1-5	0.54	5	1.67	5	**0.04**	5	0.14	5
2-5	0.57	5	1.85	5	**0.13**	5	1.16	5
3-5	0.79	5	2.20	5	**0.34**	5	8.66	5
4-5	0.64	5	2.06	5	**0.31**	5	10.91	5
1-10	4.40	5	37.30	5	**0.09**	5	0.24	5
2-10	8.04	5	42.63	5	**0.84**	5	4.87	5
3-10	**9.20**	5	49.80	5	15.03	5	124.43	4
4-10	**5.31**	5	22.84	5	23.33	5	288.80	1
1-15	34.52	5	157.79	5	**0.11**	5	0.23	5
2-15	76.36	5	294.88	1	**5.37**	5	26.72	5
3-15	**67.92**	5	268.61	3	105.00	4	300.00	0
4-15	**66.98**	5	244.42	4	232.62	4	300.00	0

Table 4.3: Average time *avg* over the 5 generated instances and the number of instances solved in a given time (300 s) for each combination of the number of machines M and number of jobs J

$J \in \{10, 15\}$ being on average nearly 3 times faster than the global MILP model for $J = 15$, $M \in \{3, 4\}$.

As for the approaches which are using CP models – the branch-and-price is also faster when the number of machines is high (3 or 4), while the global model wins when the number of machines is low (1 or 2). However, compared to the respective approaches based on MILP models, CP seems to be a lot worse. Specifically, the branch-and-price based on CP is on average approximately 4.67 times slower than BP MILP (averaged over all instances). As for the global models, the number is even higher – the global CP model is on average 12.39 times slower than the global MILP model.

It is hard to say why CP models do not perform well. It may be caused by the complicated objective function or by the structure of the problem itself; maybe, some of the constraints were not formulated in the most efficient way. Anyway, data recorded during the solution procedure indicated that for some of the larger instances, the optimal solution was found relatively quickly by the global CP model, but the solver failed to prove its optimality. So it may be that the Gurobi solver is just a way faster in optimality proving for this type of problem.

One of the important statistics of a branch-and-price procedure are the number of visited nodes of the branching tree and the number of generated patterns. These are listed in Table 4.5. Notice that the number of generated patterns increases with the increasing J, but is comparable when J is fixed and M increases. Also note that many of the smaller instances were solved in the root note. That is very desirable behaviour because branching implies more patterns to be generated (as both of the branches need to be solved or pruned). Remember that to get a single pattern – a pricing problem needs to be solved; even though it is a problem with a single machine only, it is still $\mathcal{NP}$-hard.

Sometimes, however, branching is inevitable. To show its influence, the solving times of instances 3-15 are listed in Table 4.4. It is somehow straightforward that more generated patterns will usually imply longer solving times (but not necessarily, it depends on the used prices too; usually, patterns with more jobs need longer time to be solved).

		BP MILP			BP CP		
M-J	id	#n	#p	*time*	#n	#p	*time*
3-15	1	4	89	37.47	1	64	191.81
	2	10	161	137.86	12	111	300.00
	3	7	151	86.14	5	113	272.95
	4	1	67	33.36	1	68	300.00
	5	6	108	62.74	6	95	278.27

Table 4.4: Numbers of generated patterns #p and nodes #n in contrast with the solution times *time*

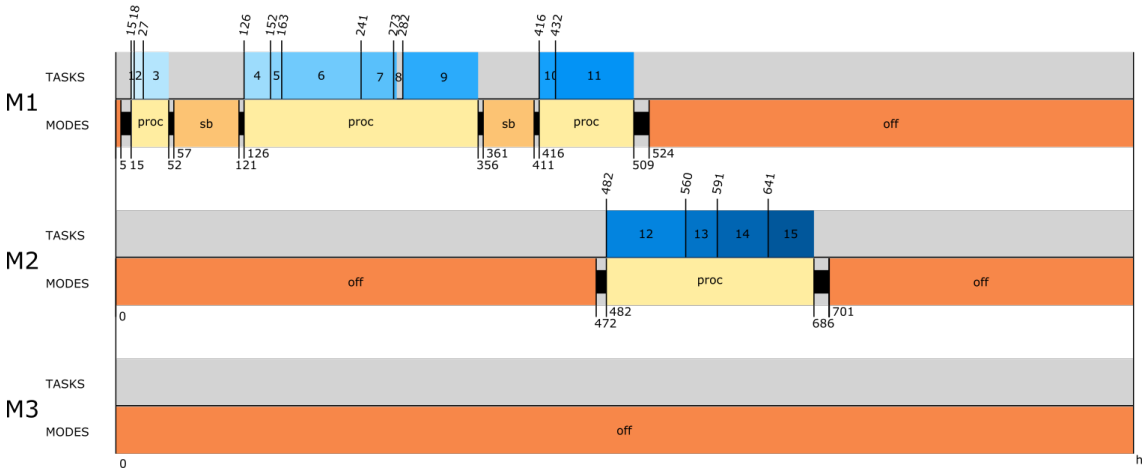


Figure 4.5: The optimal solution to instance $M = 3$, $J = 15$, id $= 4$

Note that for several instances, the numbers of patterns/nodes generated by a branch-and-price algorithm are different for MILP and CP. That is caused by the possible existence of multiple optimal solutions to the pricing problem, which generate different patterns (branching trees). What is important is that both approaches generate a similar number of patterns. Worse results of CP are caused by longer solving time per a single pattern.

The optimal solution to the instance 3-15, id $= 4$ is shown in Figure 4.5, illustrating the utilization of the energy saving mode (standby).

M-*J*	id	BP MILP		BP CP		*M*-*J*	id	BP MILP		BP CP	
		#n	#p	#n	#p			#n	#p	#n	#p
1-5	1	1	9	1	9	3-10	1	1	29	1	32
	2	1	9	1	9		2	1	30	1	29
	3	1	11	1	11		3	5	53	15	91
	4	1	9	1	9		4	4	43	4	43
	5	1	11	1	11		5	7	73	7	67
2-5	1	1	10	1	10	4-10	1	1	29	1	29
	2	1	11	1	10		2	1	28	1	22
	3	1	8	1	8		3	6	43	3	32
	4	1	9	1	9		4	1	31	1	32
	5	1	13	1	13		5	1	30	1	27
3-5	1	1	9	1	9	1-15	1	1	68	1	68
	2	1	10	1	11		2	1	90	1	97
	3	1	11	1	11		3	1	74	1	70
	4	3	16	3	15		4	1	96	1	74
	5	1	10	1	10		5	1	97	1	67
4-5	1	1	10	1	9	2-15	1	17	219	1	46
	2	1	10	1	10		2	5	151	7	120
	3	1	10	1	9		3	1	86	1	60
	4	1	13	1	13		4	1	101	1	86
	5	1	12	1	12		5	1	76	1	76
1-10	1	1	41	1	38	3-15	1	4	89	1	64
	2	1	27	1	29		2	10	161	12	111
	3	1	26	1	28		3	7	151	5	113
	4	1	34	1	40		4	1	67	1	68
	5	1	38	1	46		5	6	108	6	95
2-10	1	1	43	1	45	4-15	1	1	75	1	80
	2	1	31	1	33		2	23	249	1	53
	3	1	43	1	39		3	6	108	3	76
	4	1	43	1	42		4	1	67	1	66
	5	4	54	1	45		5	1	64	1	56

Table 4.5: Numbers of nodes #n and patterns #p for individual instances; gray cells correspond to the instances, which were not solved optimally in the time limit

4.5 Experiment 3: Multiple processing modes

For this experiment, the transition graph was changed to have two processing modes. Furthermore, data were generated in a more systematic way as described in Section 4.5.1.

4.5.1 Settings

The transition graph selected for this experiment is shown in Figure 4.6. It has four vertices – one representing an initial state, in which the machines are turned off and have zero power consumption, one representing a standby mode and two for processing modes. The first processing mode is labelled as "fast", while the second one is "slow". It is assumed, that individual jobs can be processed in both of these modes; however, processing in the slow mode will take 2 times more time (but half of the power) than processing in the fast mode. The transition times/costs were inspired by real furnaces – transition to a processing mode is relatively fast, but consumes a lot of energy (as the machine is heating); on the other hand, transition from a processing state to the standby/off mode does not consume much energy (as the machine is cooling down), but it takes more time. Note, that "fast" represents a high-temperature state of the machine, hence it consumes more power than "slow", for which the temperature is lower. Real resources, which may be described similarly, are, for example, cement kilns or chemical processing plants, where chemical substances need to dry. The minimal time spent in slow/fast and standby mode were set 20 and 10 time units, respectively, to simulate technological restrictions.

As for the parameters of the jobs, the generation scheme used for this experiment was inspired by [50], where authors review multiple data-generation approaches for scheduling applications and propose a universal generation scheme.

Parameters of the jobs were generated in the following way:

$$\mathrm{p}_{j,\mathrm{fast}} \sim U(1,100), \quad \forall j \in \mathcal{J}, \tag{4.7}$$

$$\mathrm{r}_1 = \mathrm{t}_{\mathrm{off,slow}}, \tag{4.8}$$

$$\mathrm{r}_j \sim \mathrm{r}_{j-1} + Exp\left(\beta_1 \cdot \frac{1}{|\mathcal{J}|} \sum_{j' \in \mathcal{J}} \mathrm{p}_{j',\mathrm{fast}}\right), \quad \forall j \in \{2,\ldots,J\}, \tag{4.9}$$

$$\mathrm{d}_j = \mathrm{r}_j + \beta_2 \cdot \mathrm{p}_{j,\mathrm{fast}}, \quad \forall j \in \mathcal{J}. \tag{4.10}$$

Processing times of the individual jobs are generated independently from uniform integer distribution. Release time of the first job is set to $\mathrm{t}_{\mathrm{off,slow}}$, which is the shortest possible time needed to turn a machine from the "off" state to a processing state. Release times of the other jobs are generated as a release time of the previous job plus random value generated from the exponential distribution $Exp(\lambda)$, where λ denotes a *scale* parameter (which is reciprocal of the *rate* parameter). Parameter λ in fact represents an expected

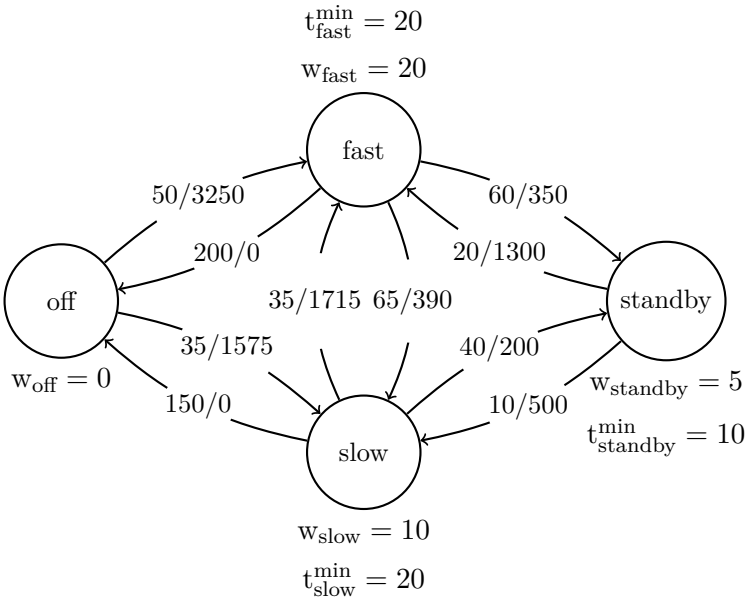


Figure 4.6: The transition graph used for Experiment 3; edges are labelled by the transition times/transition costs

value of the exponential distribution. It is set to $\beta_1 \cdot \frac{1}{|\mathcal{J}|} \sum_{j' \in \mathcal{J}} p_{j',\text{fast}}$, so the release times are generated in such a way, that the expected length of an interval between release times of jobs j and $(j+1)$ is β_1 times the average processing time, where $\beta_1 > 0$ is a parameter. Deadlines are set simply to β_2 multiples of the "fast" processing times, where $\beta_2 \geq 1$ is also a parameter. Allowing β_1 and β_2 to vary creates a wide variety of instances.

For this experiment, a number of transitions was fixed to 6 ($i_{\max} = 7$), five to fifteen jobs were generated, $J \in \{5, 10, 15\}$, to be scheduled on M machines, $M \in \{1, 2, 4\}$. Parameters were set to $\beta_1 \in \{0.7, 1.0, 1.5\}$ and $\beta_2 \in \{1.5, 2.0, 2.5\}$. Note that for $\beta_2 = 1.5$, jobs can be processed only in the "fast" mode, for $\beta_2 = 2.0$, the "slow" mode can be used, but then a job has to start exactly at its release time and finally for $\beta_2 = 2.5$, the "slow" mode allows multiple possible start times. Scheduling horizon h was set to $\max_j d_j + t_{\text{fast,off}}$. Five random instances (parameters of the jobs) were generated for each combination of J, β_1 and β_2. The same instances were used for different numbers of M to to illustrate its influence on the behaviour of the proposed approaches. The initial and the terminal mode are set to *off*-mode. A timeout was set to 300 seconds.

4.5.2 Results

The average processing times and the number of solved instances for each combination of parameters β_1 and β_2 are listed in Table 4.6. A more detailed view can be seen in Appendix B.

For the instances with $\beta_2 = 1.5$, for which the "slow" mode cannot be used, the branch-and-price approaches performed well, solving all ($\beta_1 = 1.5$) or nearly all ($\beta_1 \in \{0.7, 1.0\}$) of the

		BP MILP		BP CP		Global MILP		Global CP	
#feas	β_1-β_2	*avg*	#solved	*avg*	#solved	*avg*	#solved	*avg*	#solved
14	0.7-1.5	**15.63**	**44**	24.52	43	24.47	42	63.33	37
28	0.7-2.0	**64.29**	**38**	84.56	36	74.11	37	103.96	32
31	0.7-2.5	90.71	35	124.55	30	**55.62**	**38**	94.74	34
20	1.0-1.5	32.54	**43**	38.77	43	**28.07**	42	73.08	35
29	1.0-2.0	70.09	36	95.19	36	**66.03**	**38**	99.75	33
34	1.0-2.5	87.50	**35**	130.04	27	**73.06**	**35**	105.88	33
17	1.5-1.5	**20.13**	**45**	25.21	43	27.88	42	52.46	38
35	1.5-2.0	76.30	35	99.51	35	**74.77**	**37**	98.02	33
36	1.5-2.5	92.27	**34**	128.20	30	**84.96**	**34**	110.53	31

Table 4.6: The average solution times *avg* and the numbers of solved instances for fixed combinations of parameters β_1 and β_2 for all of the tested approaches; #feas denotes a number of feasible instances (maximally 45)

instances in a given time (300 s). The global model using MILP formulation managed to solve slightly fewer instances than both of the branch-and-price approaches, while the CP global model solved the least number of instances having the highest average time.

For instances with $\beta_2 \in \{2, 2.5\}$, the situation is not that simple. The performance of the branch-and-price with MILP pricing model is comparable to the global MILP model, but the global model has slightly better average times. Closer inspection (see Appendix B) shows, that the global model beats the branch-and-price when the number of machines is low ($M \in \{1, 2\}$), while the branch-and-price performs better when the number machines (and jobs) is higher ($M = 4$, $J = 15$). Similar behaviour can be noticed, when comparing a branch-and-price using CP pricing model to the global CP model. The global model systematically outperforms the branch-and-price when $M \in \{1, 2\}$, while being significantly worse when $M = 4$; compare 33.04 seconds (BP CP) with 240.74 seconds (global CP) for $\beta_1 = 1.0$, $\beta_2 = 1.5$, $M = 4$, $J = 10$. This behaviour was expected, as the main purpose of using branch-and-price is to reduce symmetries arising due to multiple machines.

Comparison of objective values

Most of the instances with $\beta_2 \geq 2.0$ were not solved optimally in a given time for high numbers of jobs and resources. In order to compare the quality of generated solutions, ratio of the found objective value to the best objective value across all of the approaches can be computed for each generated instance. Table 4.7 shows the ratios averaged over all the instances for fixed combinations of β_1 and β_2. Results are positive numbers greater than or equal to one, where 1 would mean, that the solution method provided the best objective values out of all tested methods. Number 1.030 would mean, that the objective values of the method were on average 3 % worse than the values found by other methods.

β_1-β_2	BP MILP	BP CP	Global MILP	Global CP
0.7-1.5	1.0030	1.0085	**1.0002**	1.0194
0.7-2.0	1.0174	1.0191	**1.0052**	1.0202
0.7-2.5	1.0100	1.0256	1.0068	**1.0007**
1.0-1.5	1.0048	1.0050	**1.0000**	1.0041
1.0-2.0	1.0282	1.0429	**1.0004**	1.0039
1.0-2.5	1.0324	1.0289	**1.0050**	1.0097
1.5-1.5	**1.0000**	1.0026	1.0011	1.0043
1.5-2.0	1.0301	1.0316	**1.0016**	1.0036
1.5-2.5	1.0348	1.0427	**1.0030**	1.0112

Table 4.7: An average ratios (over all instances, for fixed parameters β_1 and β_2) of the objective value found by a tested approach to the best objective value across all of the approaches

It shows, that the global models, especially the global MILP model, provide solutions of better quality compared to the branch-and-price approaches. That is not unexpected as the global models solve the problem as a whole, which allows them to improve the solution iteratively while using the global knowledge about all of the constraints/variables. On the other hand, the decomposed problem is solved by repeatedly adding new patterns, which may (or may not) improve the solution. A new integer solution might appear when a node is completely solved, but if it does not appear, new branches occur and the process is repeated (possibly without providing any good integer solution for some time).

Branch-and-price: possible improvement

Thanks to the decomposition, the problem can be solved iteratively, by solving the pricing problems, which are simpler than the whole global model. However, it would still take a lot of time, if the number of patterns (pricing problems to be solved) was high. Luckily, solving the pricing problems till optima is not really necessary, as any solution (pattern) with a negative reduced cost has a potential to improve the objective value of the master problem. So, instead of solving the pricing problems till optima, it is sufficient to find the first negative-cost solution. Unfortunately, that could lead to repetitive improvements of a single pattern, converging to the optimal schedule for the pattern slowly. To avoid this, the set of jobs given by the pattern corresponding to the first negative-cost solution to the pricing problem can be optimized (looking for the optimal monoprocessor schedule for these jobs). Note, that this problem is easier (compared to the whole pricing problem) as the jobs are fixed by the pattern (while for the original pricing problem, only the preferences are given and the optimization procedure selects jobs from the whole $\mathcal{J}$).

It seems, that the proposed idea (i.e., finding the first negative-cost solution and its asso-

BP MILP (first negative reduced-cost)			
β_1-β_2	*avg*	#solved	*ratio*
0.7-1.5	**9.98**	**45**	**0.9995**
0.7-2.0	**59.42**	**39**	**0.9995**
0.7-2.5	86.18	35	1.0075
1.0-1.5	**19.92**	**43**	1.0001
1.0-2.0	71.52	36	1.0019
1.0-2.5	85.79	**35**	1.0082
1.5-1.5	**5.85**	**45**	**1.0000**
1.5-2.0	**73.44**	36	1.0024
1.5-2.5	88.82	**34**	1.0078

Table 4.8: Results for the modified branch-and-price MILP procedure finding the first solution with a negative reduced-cost and re-optimizing it; the average time *avg*, the number of solved instances and the average ratio of the objective value to the best objective value across all other approaches are shown

ciated pattern and re-optimizing the pattern) could lead to better results (faster solving times). In order to verify it, it was integrated to the branch-and-price algorithm and used to solve all of the testing instances. Only the MILP model was tested in this way, as it performed better in all of the previous experiments. The results are shown in Table 4.8. The modified method completely outperformed the original BP MILP approach. Moreover, it overcame (or was equal to) all of the other approaches on multiple occasions (these are written in bold in Table 4.8). Note, that the *ratio* is compared to all others approaches (not itself) and so can be lower than 1 if the modified approach found solution of better quality (compared to all other approaches).

Branch-and-price: patterns and nodes

Some of the important statistics of the branch-and-price algorithm are the number of generated patterns and nodes. These are shown in Table 4.9. Note, that most of the instances with 15 jobs and 1 or 2 machines were infeasible, while instances with 15 jobs and 4 machines were not solved in a given time, therefore rows with $J = 15$ do not reflect the number of nodes/patterns needed to solve the problem, but they show how many patterns were generated by the respective approach in a given time. It is apparent, that even though the improved MILP approach might need more patterns than BP approaches which solve the pricing problem till optima, it is able to generate many more patterns in a given time. Standard MILP and CP can be compared too – MILP was able to generate more patterns for $J = 15$, while results are similar for $J \in \{5, 10\}$. So it seems, that CP might need more time to generate one pattern.

		BP MILP		BP CP		BP MILP *first neg.*	
β_1-β_2	J	#nodes	#patterns	#nodes	#patterns	#nodes	#patterns
0.7-1.5	5	1.00	10.33	1.00	10.11	1.00	10.50
	10	1.93	21.60	2.00	20.40	1.47	15.00
	15	1.13	63.80	1.40	60.20	1.40	76.00
0.7-2.0	5	1.00	10.91	1.00	10.82	1.00	12.18
	10	4.20	59.40	5.07	62.80	4.27	80.50
	15	1.00	46.42	1.00	42.00	1.80	158.86
0.7-2.5	5	2.46	13.50	2.60	13.57	2.33	13.42
	10	10.40	88.80	8.80	79.50	5.00	102.60
	15	1.27	54.60	1.00	41.70	1.67	149.70
1.0-1.5	5	1.27	11.45	1.27	11.45	1.27	12.55
	10	1.00	27.88	1.00	27.63	1.00	36.75
	15	3.00	82.57	1.93	59.57	3.33	130.29
1.0-2.0	5	2.47	14.00	2.47	14.00	2.33	16.45
	10	2.33	43.18	2.33	42.54	2.33	67.54
	15	2.73	61.33	1.00	41.44	5.33	171.67
1.0-2.5	5	1.00	10.60	1.00	10.87	1.00	13.00
	10	6.40	82.10	4.80	63.40	5.33	110.00
	15	1.00	35.00	1.00	38.30	1.00	134.10
1.5-1.5	5	1.00	8.40	1.00	8.10	1.00	9.20
	10	1.00	29.67	1.00	26.89	1.00	36.56
	15	1.00	45.50	1.00	36.33	1.00	41.50
1.5-2.0	5	1.00	12.47	1.00	12.47	1.00	14.40
	10	1.60	40.00	1.67	38.55	1.67	54.54
	15	1.00	44.30	1.00	41.40	1.13	138.1
1.5-2.5	5	1.53	12.07	1.53	12.07	1.53	12.85
	10	3.47	56.67	2.87	46.41	3.47	71.33
	15	1.00	30.91	1.00	28.36	1.00	110.45

Table 4.9: The average numbers of nodes and patterns for different branch-and-price formulations; BP MILP *first neg.* represents the improved branch-and-price which uses the re-optimized first negative-cost solution of the pricing problem

		M		
		1	2	4
	5	25.46 %	46.56 %	73.31 %
J	10	22.56 %	43.90 %	69.11 %
	15	20.87 %	40.33 %	66.97 %

Table 4.10: Average objective cost savings compared to the worst-case scenario for J jobs and M machines

Energy savings

Without the real industrial data, it is hard to tell, how big cost savings could be achieved in a real life. In order to show some comparison, the best objective values found in this experiment were compared with the *worst-case* scenario, where all the machines are turned to the fast mode at the beginning of the scheduling horizon and are all turned off at the end. The worst-case scenario is not totally unrealistic as in some of the production facilities the machines are scheduled in such a way. It is not really efficient, but is it comfortable for the factory workers and the machines are not strained by switching between different modes (as the repeated heating and cooling could burden the machines if overused).

The results are shown in Table 4.10. The best solution over all approaches was compared to the worst case scenario for each generated instance. Afterwards, the results were averaged over all combinations of β_1, β_2. Only the feasible instances (for which some solution was found) were used. The results show a simple dependency – the more machines there are, the more costs can be saved by scheduling of machine modes.

Of course, in real life production, the savings might not be as large as shown here. The final numbers would depend on the specific transition graph and on how "underutilized" the machines would actually be.

5 Epilogue

5.1 Conclusions

The production scheduling problem with multiple parallel identical machines was addressed in this work. The goal was to save costs by changing modes (internal states) of the machines. The problem was formulated in such a way, that the modes are explicitly modelled, so the resulting schedules provide the assignment of the jobs to machines together with the specific working profiles of the machines.

To solve the problem exactly, several approaches were proposed. At first, a global model was developed. It integrates all of the restriction in a single formulation, which can be solved by standard solvers. However, due to the structure of the problem, it might be inefficient if there are many machines to be scheduled. To improve the performance and to eliminate the symmetries arising due to the parallel machines, the global model was decomposed by a Dantzig-Wolfe decomposition technique. The decomposed approach used a branch-and-price procedure to find an optimal solution. The global model and the pricing model used in the branch-and-price were both formulated as MILP and CP problems to compare two different formulation/solution frameworks. In order to compare the proposed approaches to the state of the art methods, a reference MILP model was adapted from the literature.

Several experiments were conducted to assess the performance of the proposed models. Experiment 1 demonstrated that time-indexed models cannot be used when the scheduling horizon is too long and its discretization is too dense because in such a case, the models became too large (having many constraints and variables) to be efficiently solved even by the currently best solvers (such as Gurobi or IBM CP Optimizer).

In Experiment 2, global models were compared to the branch-and-price approach. The global models were better when the number of machines was low, whereas the branch-and-price was more effective when the number of machines was higher. This type of result was expected, as the used decomposition aims to reduce exactly these symmetries caused by the multi-machine characteristics of the problem, which might slow down the solution process of the global models. The surprising result was that MILP models outperformed CP models by a wide margin. That might be caused by the complicated objective function used in the CP models or by the solution process of the CP itself, which may be slower than the solution process of MILP for this particular problem.

Finally, Experiment 3 illustrated the generality of the proposed models, as multiple processing modes were assumed. Various benchmark instances were generated to test the efficiency of the global models and the branch-and-price procedure. The branch-and-price algorithm was slightly modified, to find and re-optimize only the first negative-solution to the pricing problem in each iteration. This modification proved to be significantly better than the original one, as the time spent on the full optimization of each pricing problem is not negligible and can be used to generate more patterns instead.

5.2 Future work

It would be naive to think, that the problem is now solved. The exact approaches allow us to investigate the structure of the problem and provide us with the optimal solutions to small instances. However, for the real production scheduling problems, tenths, hundredths or even more jobs need to be scheduled over the scheduling horizon. For now, it would be impossible to employ the proposed approaches to solve such problems. Therefore, the next necessary step is to develop an efficient heuristic, which would be able to solve large problems in a reasonable time.

As mentioned, the used decomposition has a relation to a Lagrangian relaxation, and so it may be possible to utilize it to get better lower bounds, which could be compared to the heuristic solutions, for example.

Another matter to be addressed relates to the input data. Even though the benchmark problems used in this work were inspired by real production processes, the transition graphs were created artificially, and the parameters of the jobs were generated randomly. It would be really helpful to obtain a real data from the production as the practical problems do not have a random structure and so it might be possible to adapt the algorithms to solve the real problems (instead of the general ones) fast.

Dantzig-Wolfe decomposition was successfully used to simplify the structure of the problem. It would be interesting to investigate if the decomposition could be even used for a monoprocessor problem. The proposed models schedule the individual intervals, so it might be possible to generate the intervals by the pricing problem, while the master problem would arrange them together to form the complete schedule.

All in all, only the first step in optimizing of the underutilized machines schedules was done; a long road lies ahead of the researchers, but it is worth the effort as the energy savings can be achieved with minimal investments afterwards.

References

[1] E. R. Office, "Yearly report on the operation of the czech electricity grid 2016." https://www.eru.cz/en/elektrina/statistika-a-sledovani-kvality/rocni-zpravy-o-provozu, 2017. [Online; accessed 10-April-2018].

[2] BP, "Statistical review of world energy 2017." https://www.bp.com/en/global/corporate/energy-economics/statistical-review-of-world-energy.html, 6 2017. [Online; accessed 9-April-2018].

[3] C. Gahm, F. Denz, M. Dirr, and A. Tuma, "Energy-efficient scheduling in manufacturing companies: A review and research framework," *European Journal of Operational Research*, vol. 248, pp. 744–757, 2 2016.

[4] G. Mouzon, M. B. Yildirim, and J. Twomey, "Operational methods for minimization of energy consumption of manufacturing equipment," *International Journal of Production Research*, vol. 45, pp. 4247–4271, 5 2007.

[5] S. Mitra, I. E. Grossmann, J. M. Pinto, and N. Arora, "Optimal production planning under time-sensitive electricity prices for continuous power-intensive processes," *Computers & Chemical Engineering*, vol. 38, pp. 171–184, 3 2012.

[6] J. Dušek, "Návrh úpravy řízení výrobní linky s ohledem na snížení její spotřeby," Master's thesis, Czech Technical University in Prague, the Czech republic, 6 2016.

[7] I. E. Agency, "Co2 emissions from fuel combustion; statistics 2017; highlights." https://www.iea.org/publications/freepublications/publication/CO2EmissionsfromFuelCombustionHighlights2017.pdf, 10 2017. [Online; accessed 15-April-2018].

[8] M. Rager, C. Gahm, and F. Denz, "Energy-oriented scheduling based on evolutionary algorithms," *Computers & Operations Research*, vol. 54, pp. 218–231, 2 2015.

[9] F. Shrouf, J. Ordieres-Meré, A. García-Sánchez, and M. Ortega-Mier, "Optimizing the production scheduling of a single machine to minimize total energy consumption costs," *Journal of Cleaner Production*, vol. 67, pp. 197–207, 5 2014.

[10] M. Selmair, T. Claus, F. Herrmann, A. Bley, and M. Trost, "Job shop scheduling with flexible energy prices," 6 2016. European Conference on Modelling and Simulation.

[11] X. Gong, T. D. Pessemier, W. Joseph, and L. Martens, "A generic method for energy-

efficient and energy-cost-effective production at the unit process level," *Journal of Cleaner Production*, vol. 113, pp. 508–522, 2 2016.

[12] K.-T. Fang and B. M. T. Lin, "Parallel-machine scheduling to minimize tardiness penalty and power cost," *Computers & Industrial Engineering*, vol. 64, pp. 224–234, 1 2013.

[13] E. K. Boukas, A. Haurie, and F. Soumis, "Hierarchical approach to steel production scheduling under a global energy constraint," *Annals of Operations Research*, vol. 26, pp. 289–311, 1 1991.

[14] J.-Y. Moon, K. Shin, and J. Park, "Optimization of production scheduling with time-dependent and machine-dependent electricity cost for industrial energy efficiency," *The International Journal of Advanced Manufacturing Technology*, vol. 68, pp. 523–535, 9 2013.

[15] C. Artigues, P. Lopez, and A. Haït, "The energy scheduling problem: Industrial case-study and constraint propagation techniques," *International Journal of Production Economics*, vol. 143, pp. 13–23, 5 2013.

[16] A. Che, S. Zhang, and X. Wu, "Energy-conscious unrelated parallel machine scheduling under time-of-use electricity tariffs," *Journal of Cleaner Production*, vol. 156, pp. 688–697, 7 2017.

[17] Z. Hanzálek and P. Šůcha, "Scheduling." `https://rtime.felk.cvut.cz/~hanzalek/KO/sched_e.pdf`, 2018. [Online; accessed 17-May-2018].

[18] G. Gamrath, *Generic Branch-Cut-and-Price*. PhD thesis, Technischen Universität Berlin, Germany, 2010.

[19] D. Feillet, "A tutorial on column generation and branch-and-price for vehicle routing problems," *4OR*, vol. 8, pp. 407–424, 12 2010.

[20] M. Akella, S. Gupta, and A. Sarkar, "Branch and price, column generation for solving huge integer programs." `https://www.acsu.buffalo.edu/~nagi/courses/684/price.pdf`, 2004. [Online; accessed 15-April-2018].

[21] G. Optimization, "Mixed-integer programming (mip) - a primer on the basics." `http://www.gurobi.com/resources/getting-started/mip-basics`, 2018. [Online; accessed 17-April-2018].

[22] G. B. Dantzig and P. Wolfe, "Decomposition principle for linear programs," *Operations Research*, vol. 8, pp. 101–111, 2 1960.

[23] J. Desrosiers and M. E. Lübbecke, *A Primer in Column Generation*, pp. 1–32. Boston, MA: Springer, Boston, MA, 2005.

[24] T. Werner, "Optimization." `"https://cw.fel.cvut.cz/old/_media/courses/`

a4b33opt/opt.pdf", 2018. [Online; accessed 19-April-2018].

[25] L. G. Khachiyan, "Polynomial algorithms in linear programming," *USSR Computational Mathematics and Mathematical Physics*, vol. 20, no. 1, pp. 51–68, 1980.

[26] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, pp. 373–395, 12 1984.

[27] G. B. Dantzig, A. Orden, and P. Wolfe, "The generalized simplex method for minimizing a linear form under linear inequality restraints," *Pacific Journal of Mathematics*, vol. 5, pp. 183–195, 6 1955.

[28] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, vol. B of *24*. Springer-Verlag Berlin Heidelberg, 1 ed., 1 2003.

[29] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, pp. 497–520, 7 1960.

[30] R. C E Gilmore and R. Gomory, "A linear programming approach to the cutting stock problem i," *Operations Research*, vol. 9, pp. 849–859, 12 1961.

[31] G. B. Dantzig and P. Wolfe, "The decomposition algorithm for linear programs," *Econometrica*, vol. 29, pp. 767–778, 10 1961.

[32] A. Schrijver, *Theory of Linear and Integer Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1986.

[33] M. E. Lübbecke, *Column Generation*. American Cancer Society, 2011.

[34] C. Lemaréchal, "The omnipresence of lagrange," *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, vol. 1, pp. 7–25, 3 2003.

[35] T. L. Magnanti, J. F. Shapiro, and M. H. Wagner, "Generalized linear programming solves the dual," *Management Science*, vol. 22, pp. 1195–1203, 7 1976.

[36] D. Huisman, R. Jans, M. Peeters, and A. P. Wagelmans, *Combining Column Generation and Lagrangian Relaxation*, pp. 247–270. Boston, MA: Springer US, 2005.

[37] R. Václavík, A. Novák, P. Šůcha, and Z. Hanzálek, "Accelerating the branch-and-price algorithm using machine learning," *European Journal of Operational Research*, 2017. under review.

[38] F. Vanderbeck, "Branching in branch-and-price: a generic scheme," *Mathematical Programming*, vol. 130, pp. 249–294, 12 2011.

[39] V. Chvátal, *Linear Programming*. Series of books in the mathematical sciences, W.H. Freeman, 1983.

[40] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen, "Stabilized column gener-

ation," *Discrete Mathematics*, vol. 194, pp. 229–237, 1 1999.

[41] O. Briant, C. Lemaréchal, P. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck, "Comparison of bundle and classical column generation," *Mathematical Programming*, vol. 113, pp. 299–344, 6 2008.

[42] A. Bettinelli, A. Ceselli, and G. Righini, "A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows," *Transportation Research Part C: Emerging Technologies*, vol. 19, no. 5, pp. 723–740, 2011.

[43] C. Archetti, N. Bianchessi, and A. Hertz, "A branch-and-price algorithm for the robust graph coloring problem," *Discrete Applied Mathematics*, vol. 165, pp. 49–59, 2014. 10th Cologne/Twente Workshop on Graphs and Combinatorial Optimization (CTW 2011).

[44] R. Barták, "Constraint programming." `http://kti.ms.mff.cuni.cz/~bartak/podminky/index.html#kontakt`, 2018. [Online; accessed 6-May-2018].

[45] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím, "Ibm ilog cp optimizer for scheduling," *Constraints*, vol. 23, pp. 210–250, 4 2018.

[46] P. Laborie, "Introduction to cp optimizer for scheduling." `http://icaps17.icaps-conference.org/tutorials/T3-Introduction-to-CP-Optimizer-for-Scheduling.pdf`, 2017. [Online; accessed 6-May-2018].

[47] F. Vanderbeck, "On dantzig-wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm," *Operations Research*, vol. 48, pp. 111–128, 1 2000.

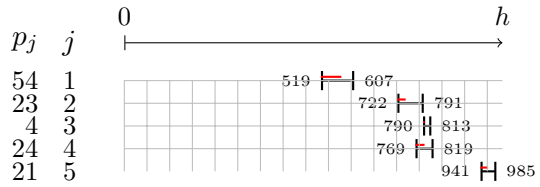
[48] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. New York, NY, USA: Wiley-Interscience, 1988.

[49] M. E. Lübbecke and J. Desrosiers, "Selected topics in column generation," *Operations Research*, vol. 53, pp. 1007–1023, 11 2005.

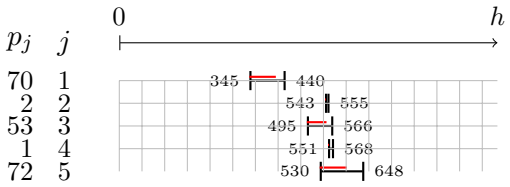
[50] N. G. Hall and M. E. Posner, "Generating experimental data for computational testing with machine scheduling applications," *Operations Research*, vol. 49, no. 6, pp. 854–865, 2001.

Appendix

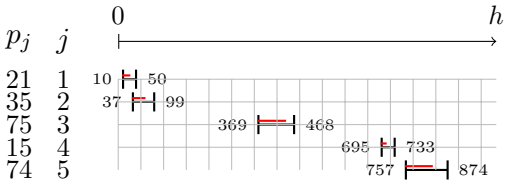
A Instances generated for Experiment 2



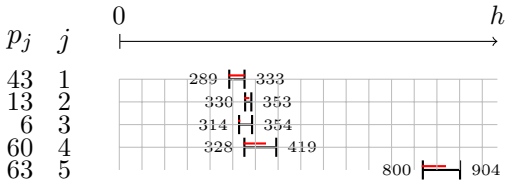
$M = 1$, $J = 5$, id $= 0$



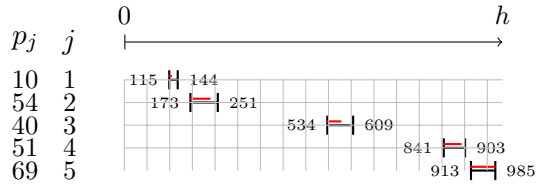
$M = 1$, $J = 5$, id $= 1$



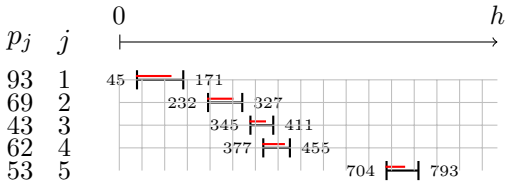
$M = 1$, $J = 5$, id $= 2$



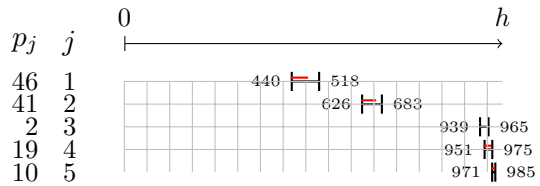
$M = 1$, $J = 5$, id $= 3$



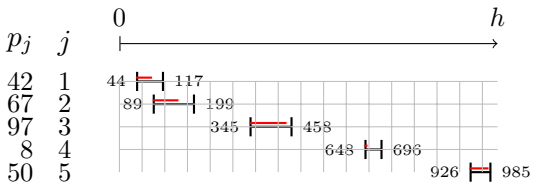
$M = 1$, $J = 5$, id $= 4$



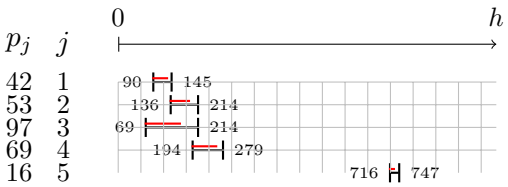
$M = 2$, $J = 5$, id $= 0$



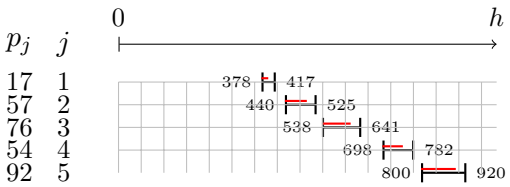
$M = 2$, $J = 5$, id $= 1$



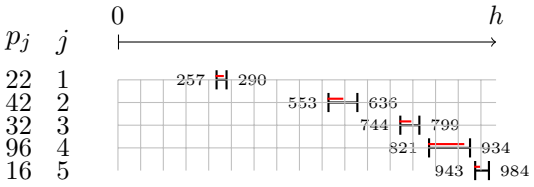
$M = 2$, $J = 5$, id $= 2$



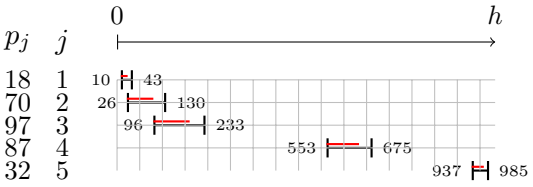
$M = 2$, $J = 5$, id $= 3$



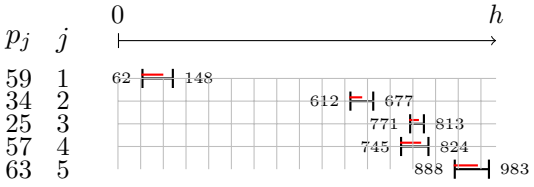
$M = 2$, $J = 5$, id $= 4$



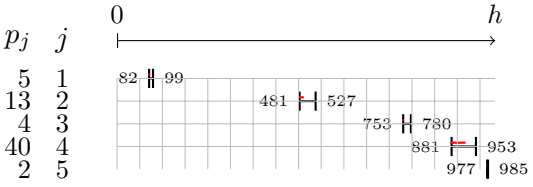
$M = 3,\ J = 5,\ \mathrm{id} = 0$



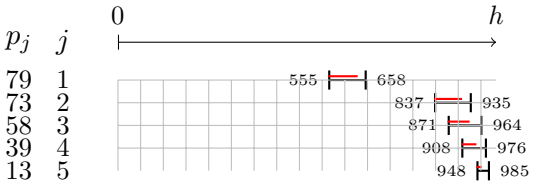
$M = 3,\ J = 5,\ \mathrm{id} = 1$



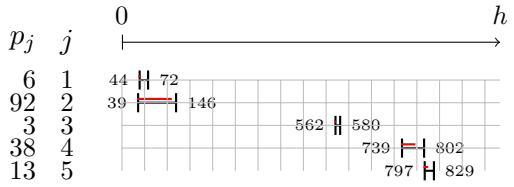
$M = 3,\ J = 5,\ \mathrm{id} = 2$



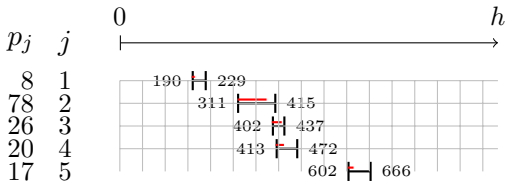
$M = 3,\ J = 5,\ \mathrm{id} = 3$



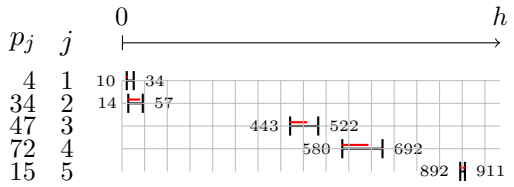
$M = 3,\ J = 5,\ \mathrm{id} = 4$



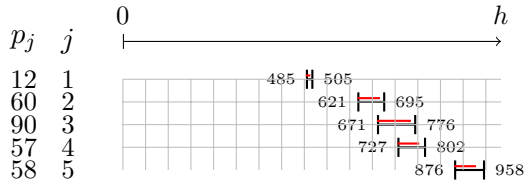
$M = 4,\ J = 5,\ \mathrm{id} = 0$



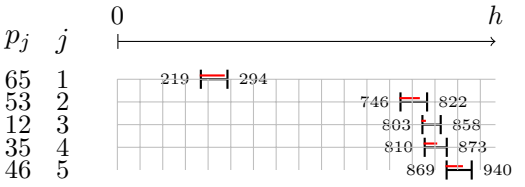
$M = 4,\ J = 5,\ \mathrm{id} = 1$



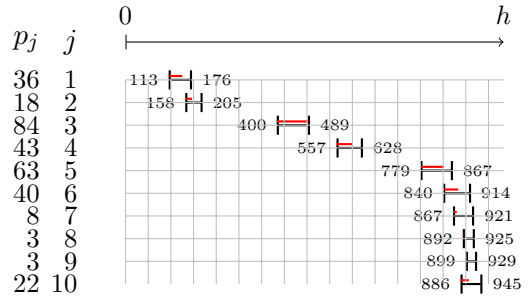
$M = 4,\ J = 5,\ \mathrm{id} = 2$



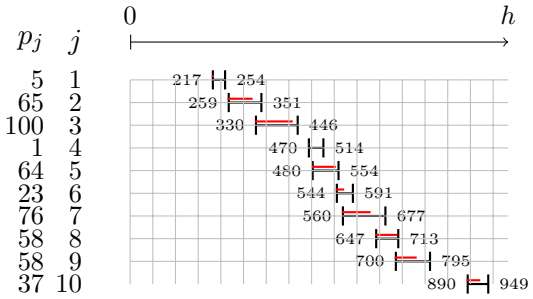
$M = 4,\ J = 5,\ \mathrm{id} = 3$



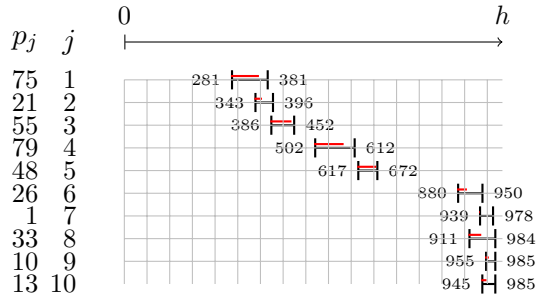
$M = 4,\ J = 5,\ \mathrm{id} = 4$



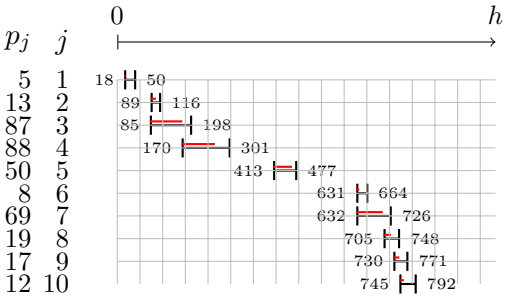
$M = 1,\ J = 10,\ \mathrm{id} = 0$



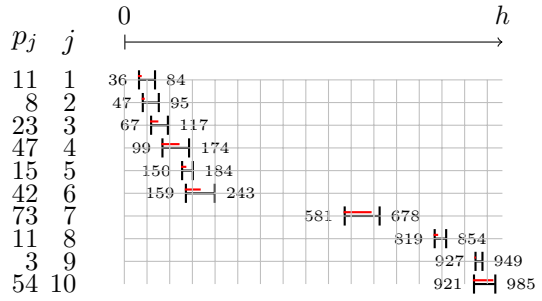
$M = 1,\ J = 10,\ \mathrm{id} = 1$



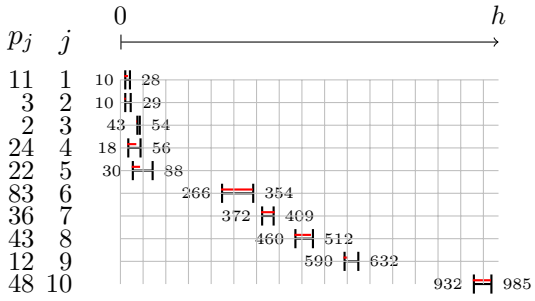
$M = 1$, $J = 10$, id $= 2$



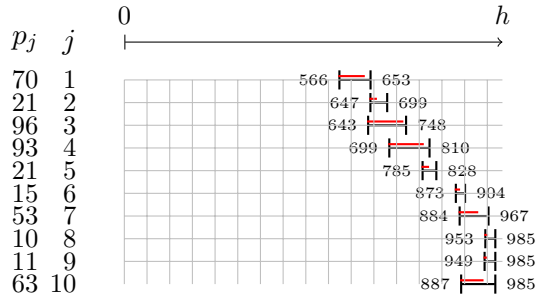
$M = 1$, $J = 10$, id $= 3$



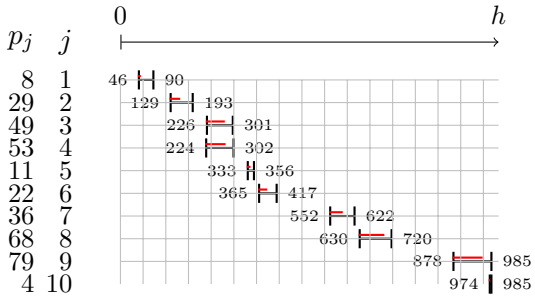
$M = 1$, $J = 10$, id $= 4$



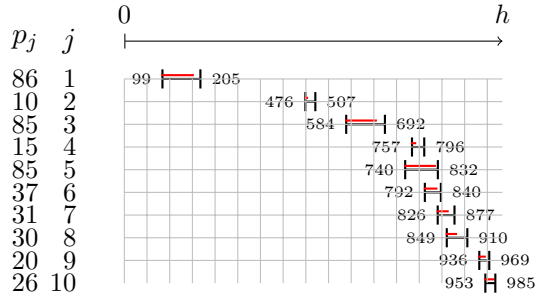
$M = 2$, $J = 10$, id $= 0$



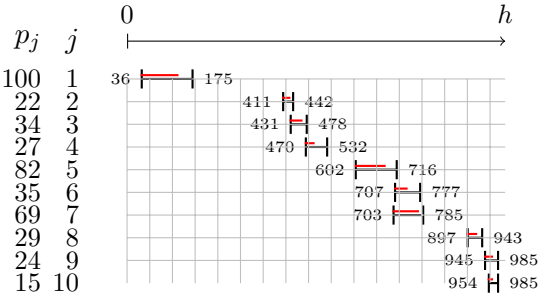
$M = 2$, $J = 10$, id $= 1$



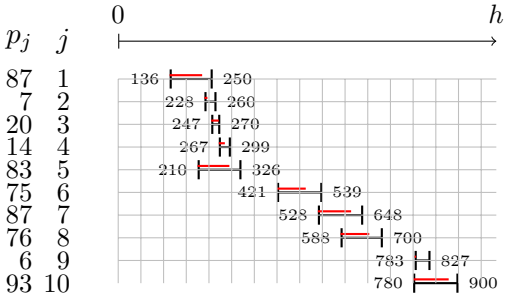
$M = 2$, $J = 10$, id $= 2$



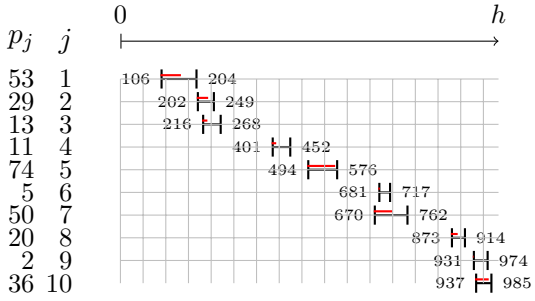
$M = 2$, $J = 10$, id $= 3$



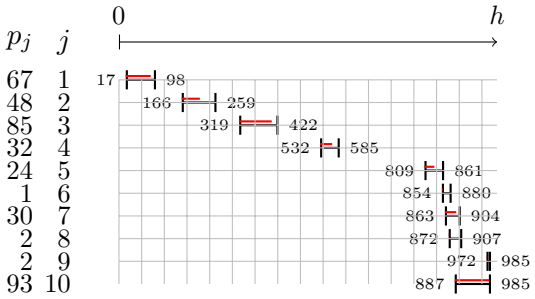
$M = 2$, $J = 10$, id $= 4$



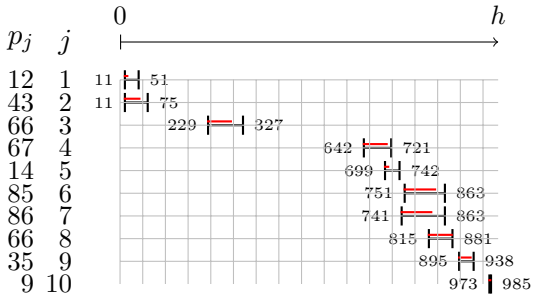
$M = 3$, $J = 10$, id $= 0$



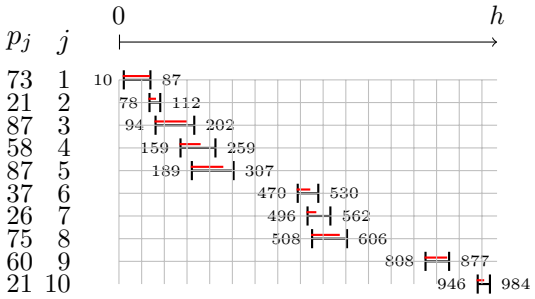
$M = 3$, $J = 10$, id $= 1$



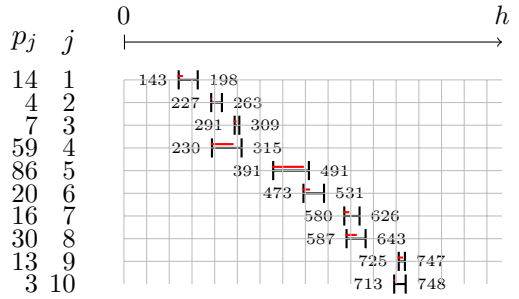
$M = 3$, $J = 10$, id $= 2$



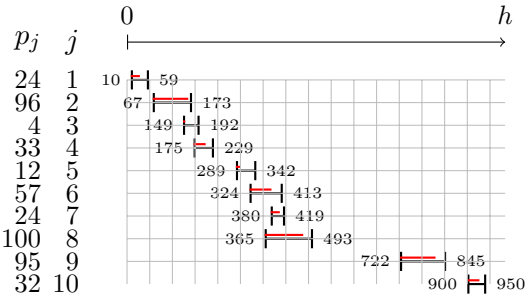
$M = 3$, $J = 10$, id $= 3$



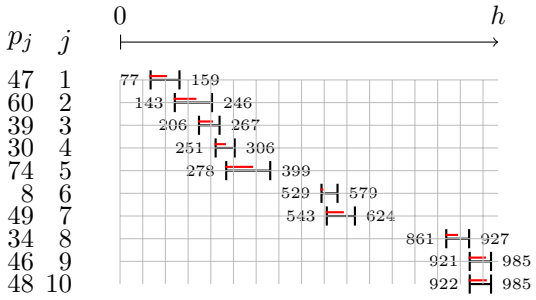
$M = 3$, $J = 10$, id $= 4$



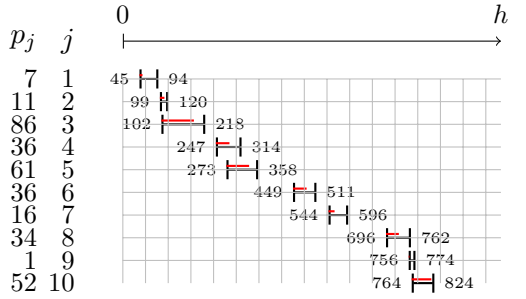
$M = 4$, $J = 10$, id $= 0$



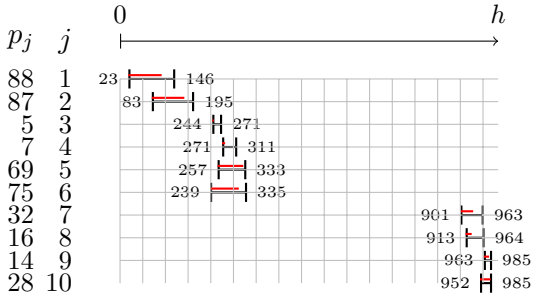
$M = 4$, $J = 10$, id $= 1$



$M = 4$, $J = 10$, id $= 2$



$M = 4$, $J = 10$, id $= 3$



$M = 4$, $J = 10$, id $= 4$

B Aggregated results of Experiment 3

Results of the Experiment 3 aggregated over the 5 randomly generated instances for each combination of parameters M, J, β_1, β_2; the number of feasible instances #feas, the average solving time *avg* and the number of solved instances #solved are listed for each approach separately.

		BP MILP		BP CP		Global MILP		Global CP	
		avg	#solved	*avg*	#solved	*avg*	#solved	*avg*	#solved
$\beta_1 = 0.7$. $\beta_2 = 1.5$									
#feas	M-J								
0	1-5	0.03	**5**	0.03	**5**	**0.00**	**5**	0.08	**5**
4	2-5	0.74	**5**	0.76	**5**	**0.20**	**5**	1.89	**5**
4	4-5	1.03	**5**	1.02	**5**	**0.56**	**5**	64.18	4
0	1-10	**0.02**	**5**	0.03	**5**	**0.02**	**5**	0.08	**5**
0	2-10	**0.03**	**5**	**0.03**	**5**	0.20	**5**	5.68	**5**
2	4-10	7.85	**5**	10.20	**5**	**3.70**	**5**	50.86	**5**
0	1-15	**0.02**	**5**	**0.02**	**5**	0.07	**5**	0.25	**5**
0	2-15	**0.04**	**5**	**0.04**	**5**	13.93	**5**	206.15	2
4	4-15	**130.92**	**4**	208.54	3	201.54	2	240.80	1
$\beta_1 = 0.7$. $\beta_2 = 2.0$									
#feas	M-J								
1	1-5	0.18	**5**	0.62	**5**	**0.06**	**5**	0.24	**5**
5	2-5	1.74	**5**	3.27	**5**	**0.49**	**5**	3.45	**5**
5	4-5	**1.45**	**5**	2.81	**5**	**1.45**	**5**	15.79	**5**
0	1-10	**0.03**	**5**	**0.03**	**5**	**0.03**	**5**	0.58	**5**
5	2-10	89.18	**5**	177.72	4	**25.52**	**5**	45.83	**5**
5	4-10	**66.02**	**5**	156.56	4	122.12	4	286.43	1
0	1-15	**0.03**	**5**	**0.03**	**5**	0.10	**5**	0.21	**5**
2	2-15	**120.02**	**3**	**120.02**	**3**	217.24	**3**	283.14	1
5	4-15	300.00	0	300.00	0	300.00	0	300.00	0
$\beta_1 = 0.7$. $\beta_2 = 2.5$									
#feas	M-J								
3	1-5	0.96	**5**	3.54	**5**	**0.08**	**5**	0.39	**5**
5	2-5	3.09	**5**	7.53	**5**	**0.88**	**5**	3.30	**5**
5	4-5	3.71	**5**	7.85	**5**	**2.08**	**5**	18.23	**5**
0	1-10	**0.03**	**5**	**0.03**	**5**	0.24	**5**	1.22	**5**
5	2-10	90.86	**5**	227.50	3	**8.29**	**5**	47.65	**5**
5	4-10	117.69	**5**	274.47	2	**97.50**	4	300.00	0
0	1-15	0.03	**5**	**0.02**	**5**	0.10	**5**	2.10	**5**

4	2-15	300.00	0	300.00	0	**91.43**	**4**	179.72	**4**
4	4-15	300.00	0	300.00	0	300.00	0	300.00	0
$\beta_1 = 1.0.\ \beta_2 = 1.5$									
#feas	*M-J*								
0	1-5	0.27	**5**	0.46	**5**	**0.02**	**5**	0.11	**5**
4	2-5	1.67	**5**	2.03	**5**	**0.24**	**5**	3.72	**5**
4	4-5	1.62	**5**	2.09	**5**	**0.71**	**5**	15.65	**5**
0	1-10	**0.02**	**5**	0.03	**5**	**0.02**	**5**	0.15	**5**
2	2-10	8.56	**5**	22.45	**5**	**3.14**	**5**	26.55	**5**
4	4-10	**11.14**	**5**	32.72	**5**	59.15	**5**	240.55	1
0	1-15	0.03	**5**	0.03	**5**	**0.02**	**5**	0.18	**5**
2	2-15	26.38	**5**	79.08	**5**	**6.64**	**5**	129.36	3
4	4-15	**162.17**	**3**	210.02	**3**	182.67	2	241.49	1
$\beta_1 = 1.0.\ \beta_2 = 2.0$									
#feas	*M-J*								
1	1-5	0.27	**5**	0.96	**5**	**0.11**	**5**	0.28	**5**
5	2-5	3.29	**5**	6.88	**5**	**0.73**	**5**	5.86	**5**
5	4-5	3.82	**5**	7.84	**5**	**2.21**	**5**	22.18	**5**
0	1-10	7.80	**5**	33.68	**5**	**0.15**	**5**	0.93	**5**
5	2-10	38.27	**5**	135.59	**5**	**19.93**	**5**	49.25	**5**
5	4-10	**37.34**	**5**	131.76	**5**	104.23	**5**	300.00	0
0	1-15	**0.03**	**5**	**0.03**	**5**	0.37	**5**	5.09	**5**
3	2-15	240.01	1	240.01	1	**189.55**	2	214.19	**3**
5	4-15	300.00	0	300.00	0	**276.99**	**1**	300.00	0
$\beta_1 = 1.0.\ \beta_2 = 2.5$									
#feas	*M-J*								
5	1-5	1.39	**5**	5.42	**5**	**0.18**	**5**	0.41	**5**
5	2-5	1.68	**5**	6.42	**5**	**0.67**	**5**	2.57	**5**
5	4-5	1.63	**5**	5.39	**5**	**1.45**	**5**	7.87	**5**
0	1-10	**0.03**	**5**	**0.03**	**5**	0.32	**5**	1.64	**5**
5	2-10	82.24	**5**	289.99	1	**12.67**	**5**	82.73	**5**
5	4-10	100.46	**5**	263.07	1	**41.91**	**5**	300.00	0
0	1-15	0.03	**5**	**0.02**	**5**	0.32	**5**	5.53	**5**
4	2-15	300.00	0	300.00	0	300.00	0	**252.21**	**3**
5	4-15	300.00	0	300.00	0	300.00	0	300.00	0
$\beta_1 = 1.5.\ \beta_2 = 1.5$									
#feas	*M-J*								
0	1-5	0.08	**5**	0.09	**5**	**0.03**	**5**	0.11	**5**
3	2-5	1.03	**5**	2.02	**5**	**0.28**	**5**	1.99	**5**
3	4-5	1.21	**5**	2.15	**5**	**0.83**	**5**	15.87	**5**
0	1-10	0.03	**5**	**0.02**	**5**	0.13	**5**	0.23	**5**

4	2-10	13.26	**5**	34.77	**5**	**5.33**	**5**	16.91	**5**
4	4-10	**12.45**	**5**	33.04	**5**	116.21	4	240.76	1
0	1-15	0.03	**5**	**0.02**	**5**	0.09	**5**	0.38	**5**
1	2-15	53.29	**5**	60.03	4	**8.01**	**5**	71.92	4
2	4-15	99.79	**5**	**94.76**	4	120.03	3	123.94	3
$\beta_1 = 1.5.\ \beta_2 = 2.0$									
#feas	M-J								
5	1-5	1.54	**5**	4.67	**5**	**0.15**	**5**	0.33	**5**
5	2-5	2.48	**5**	7.83	**5**	**0.56**	**5**	3.00	**5**
5	4-5	2.62	**5**	7.97	**5**	**1.49**	**5**	10.22	**5**
0	1-10	2.25	**5**	16.21	**5**	**0.16**	**5**	2.53	**5**
5	2-10	39.47	**5**	130.21	**5**	**13.15**	**5**	47.59	**5**
5	4-10	**38.35**	**5**	128.63	**5**	132.84	**5**	300.00	0
0	1-15	**0.02**	**5**	0.03	**5**	0.15	**5**	3.12	**5**
5	2-15	300.00	0	300.00	0	224.44	2	**215.36**	**3**
5	4-15	300.00	0	300.00	0	300.00	0	300.00	0
$\beta_1 = 1.5.\ \beta_2 = 2.5$									
#feas	M-J								
3	1-5	1.27	**5**	3.84	**5**	**0.16**	**5**	0.42	**5**
5	2-5	2.36	**5**	6.34	**5**	**0.68**	**5**	3.91	**5**
5	4-5	2.83	**5**	6.49	**5**	**1.69**	**5**	13.51	**5**
2	1-10	10.35	**5**	46.10	**5**	**0.58**	**5**	3.02	**5**
5	2-10	78.11	**5**	217.63	3	**25.77**	**5**	84.21	**5**
5	4-10	**75.51**	**5**	213.41	3	134.38	4	277.44	1
1	1-15	60.02	4	60.02	4	**1.39**	**5**	12.24	**5**
5	2-15	300.00	0	300.00	0	300.00	0	300.00	0
5	4-15	300.00	0	300.00	0	300.00	0	300.00	0

C List of abbreviations

BP	Branch-and-price
CP	Constraint Programming
FSM	Finite state machine
LP	Linear Programming
MILP	Mixed Integer Linear Programming
MP	Master Problem
RMP	Restricted Master Problem

D Contents of the attached CD

```
root
└── Energy_aware_production_scheduling
    ├── README.txt ..... information about the content
    ├── DP_Benedikt_2018.pdf ..... text of the thesis
    ├── source_text.zip ..... LaTeX source files
    └── source_code.zip ..... Python source codes
```