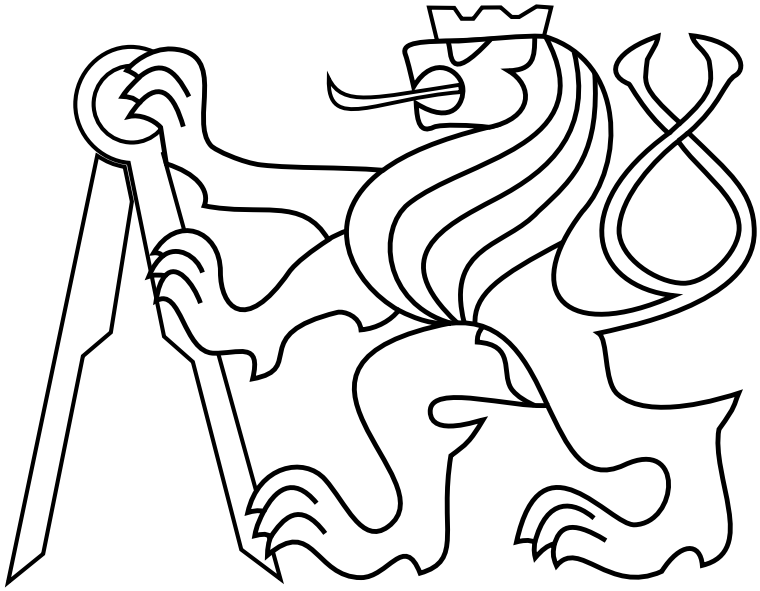
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

MASTER'S THESIS



Bc. Richard Štec

Scheduling jobs with stochastic processing time on parallel identical machines

Czech Technical University in Prague, Faculty of Electrical Engineering

Thesis supervisor: **Ing. Antonín Novák**

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Štec** Jméno: **Richard** Osobní číslo: **434784**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Kybernetická bezpečnost**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Rozvrhování úloh s neurčitou dobou trvání na paralelních identických strojích

Název diplomové práce anglicky:

Scheduling jobs with stochastic processing time on parallel identical machines

Pokyny pro vypracování:

This thesis addresses a scheduling jobs with stochastic processing time on parallel processors. The objective is to maximize probability that the makespan will not exceed a common due date. The aim is to make an attempt to propose a better algorithm that the one published in [1]. The particular objectives of the thesis are:
1) Review the existing works in the scheduling domain and analyze results described in [1].
2) For problem described in [1], devise and implement an exact scheduling algorithm based on Integer Linear Programming.
3) For the same problem propose and implement a branch-and-price algorithm.
4) Compare the devised algorithms with results published in [1].

Seznam doporučené literatury:

[1] Mohammad Ranjbar, Morteza Davari, Roel Leus:
Two branch-and-bound algorithms for the robust parallel machine scheduling problem. Computers & OR 39(7): 1652-1660 (2012)
[2] Shuwan Zhu, Wenjuan Fan, Shanlin Yang, Jun Pei, Panos M. Pardalos, Operating room planning and surgical case scheduling: a review of literature, In: Journal of Combinatorial Optimization. Online first, https://doi.org/10.1007/s10878-018-0322-6, (2019).
[3] Michael L. Pinedo. Scheduling: Theory, Algorithms, and Systems (3rd ed.). Springer Publishing Company, Incorporated. 2008.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Antonín Novák, katedra řídicí techniky FEL

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

Datum zadání diplomové práce: **29.01.2019** Termín odevzdání diplomové práce: **24.05.2019**

Platnost zadání diplomové práce: **20.09.2020**

Ing. Antonín Novák podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

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

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university thesis.

In Prague, ...

Acknowledgements

I would like to thank Antonín Novák and Přemysl Šůcha for their excellent supervision and guidance in this thesis process.

I would also like to thank my friends Michal Gabriel and José Arturo Lozano Angulo for finding a time to read this work and offer suggestions for its improvement.

Abstract

In this work we consider a stochastic parallel machine scheduling problem, where the jobs have uncertain processing time, described by a normal probability distribution. The objective is to maximize the probability, that all the jobs are completed before single common due date. To tackle this problem, we first introduce a linear approximation of the non-linear model of the problem. We also provide an exact Branch-and-Price approach and an algorithm to solve a special case of the problem with exactly two machines. Furthermore, we introduce a heuristic algorithm to give a lower bound for the objective function. Experimental results show, that the Branch-and-Price algorithm and the two-machines algorithm outperform the existing approaches.

Keywords: stochastic scheduling, β-robust scheduling, robust parallel machine scheduling problem

Abstrakt

Táto práca sa zaoberá problémom stochastického rozvrhovania, kde trvanie úlohy nie je presne známe, ale je popísané normálnym pravdepodobnostným rozdelením. Úlohou je nájsť taký rozvrh, ktorý maximalizuje pravdepodobnosť, že všetky úlohy budú dokončené pred jedným spoločným konečným termínom. Aby sme tento problém vyriešili, najprv predstavíme lineárnu aproximáciu nelineárneho modelu problému. Následne zavedieme exaktný Branch-and-Price postup a algoritmus na vyriešenie špeciálneho prípadu, kde uvažujeme len dva stroje. Taktiež predstavíme heurisitcký algoritmus, ktorý počíta spodnú medz pre kriteriálnu funkciu. Experimenty ukázali, že Branch-and-Price a algoritmus pre prípad dvoch strojov dávajú lepšie výsledky ako momentálne existujúce postupy.

Kľúčové slová: stochastické rozvrhovanie, β-robustné rozvrhovanie, robust parallel machine scheduling problem

Contents

List of Figures

1 Introduction

The problem of parallel machine scheduling arises in many real world situations. It is usually introduced with reference to production optimization [Pinedo, 2012], but finds applications in many other fields, such as rostering [Ernst *et al.*, 2004] or parallel computation [El-Rewini and Lewis, 1990]. The deterministic variant (i.e., all parameters are known and static, no disruptions are assumed, etc.) has been already extensively studied in the 20th century [Cheng and Sin, 1990]. However, it soon became clear that the application of the deterministic scheduling models does not suffice for many practical applications, since the influence of non-deterministic behavior can be too significant to ignore. This has been noted by several authors, such as [Maccarthy and Liu, 1993] or [Stoop and Wiers, 1996], and eventually gave rise to the field of stochastic scheduling.

Stochastic behavior can be observed in many factors. The machines can breakdown at any moment [Adiri *et al.*, 1989] and cannot continue operation until repaired, the supply of resources might by uncertain [Ramazan and Dimitrakopoulos, 2013], the release dates, due dates can be stochastic [Pinedo, 1983] and so on.

This work focuses on the problem of parallel machine scheduling, where the processing time of each job is not precisely known and is given by a probability distribution. The optimal solution is a schedule that maximizes the probability that all the jobs are finished before a common specified due date. A good example for the application of such scheduling model is the scheduling of operations in operation theaters, where [Stepaniak *et al.*, 2009] have shown that the processing time of a single operation can be modeled by a lognormal distribution.

1.1 Related work

There are many approaches to stochastic scheduling problems. One of the most popular, is known as the *reactive scheduling*, whose principle is a real-time response to the breakdown of a schedule. A summary of such methods is given by [Smith, 1995] or [Sabuncuoglu and Bayız, 2000].

Another approach was introduced by [Daniels and Kouvelis, 1995], who defined the term *robust scheduling*. This approach aims to create a schedule that minimizes the impact of expected stochastic effects during the execution of the schedule. [Módos *et al.*, 2017] study a single machine scheduling problem, with uncertain start time of each job. Furthermore, each job has an energy consumption assigned and the objective is to find a robust schedule that minimizes the total tardiness with respect to given energy consumption limit. They propose two exact algorithms (Branch-and-Bound and logic-based Benders decomposition) and a heuristic algorithm (tabu search). They solve instances up to 15 jobs using the exact algorithms within the given the time limit. It is also shown that the proposed heuristic tabu search algorithms solves instances with up to 100 jobs within several seconds and improves the objective by 40.2% compared to the Earliest Due Date First ordering rule. A different way to tackle the problem of creating a robust schedule is to employ the fuzzy set theory. [Dubois *et al.*, 2003] and [Wang, 2004] give overview of such methods.

A notable study is done by [Kouvelis *et al.*, 2000], who devise a Branch-and-Bound method to solve two-machine flow shop robust scheduling problem, where the processing times of jobs are uncertain. Apart from other similar studies, they do not assume *a priori* knowledge of

the distribution of the processing times, and instead describe a more general approach. They propose a branch-and-bound algorithm and a heuristic to solve the problem, managing to solve instances with up to 15 jobs, while the time average on the largest instances was 20 minutes. They also introduced a heuristic algorithm that managed to obtain the optimal solution in 95% of cases, while the run time was only several seconds. Such problems, where the distributions of the stochastic variables are not precisely known is a field of *distributionally robust optimization*. [Chang *et al.*, 2019] study the problem for multiple parallel machines, where only a moment information about the distribution of the job processing times is known. They develop an exact second order cone program and managed to solve instances with up to 100 jobs and 5 machines within several seconds.

In recent years, the problem of *β-robust scheduling* received increased attention. The term was coined by [Daniels and Carrillo, 1997] and the goal is to create a schedule that optimizes some objective function with respect to some target level β. They also introduced a method to build a β-robust schedule for a single machine problem with jobs that have uncertain processing times, while optimizing the *total flow time* criterion. They solved instances up to 20 jobs within 30 seconds on average. The same problem is also studied by [Lu *et al.*, 2012], while also considering a sequence-dependent setup times. These setup times together with the processing of jobs are uncertain and are represented using intervals. They reformulate the problem as a robust constrained shortest path problem and develop a simulated annealing framework where they propose a heuristic to solve it. To compare the performance of the heuristic they also implement an enumeration method to obtain the optimal value. They test their approach on test cases with maximum of 8 jobs and generate total of 1215 instances. In all the cases, the proposed heuristic manged to find the exact solution for the problem in less than 20 seconds. [Alimoradi *et al.*, 2016] solve *robust parallel machine scheduling problem* (RPMSP) with jobs with uncertain processing times and total flow time as criterion. They stated several theorems and proposed a specific branch-and-bound method to solve the problem. Instances with up to 45 jobs and 5 machines were solved in less than 20 minutes.

[Ranjbar *et al.*, 2012] propose two branch-and-bound algorithms to solve the problem of robust parallel machine scheduling, while maximizing the probability that all the jobs are processed before common due date δ. They also introduce methods to calculate the lower and upper bound on the objective value and a method to calculate the lower bound on the number of jobs each machine has to schedule. They solve instances with up to 5 machines and 20 jobs. Their first method, denoted as B&B1, solve the largest instances in about an hour, while the second branch-and-bound (B&B2) was unable to solve the largest instances due to limited resources. They however note that under time limit, the B&B2 algorithm is better at providing heuristic solutions. It is also shown the for the B&B1 the optimal solution is found early in the search and most of the time is spent on proving its optimality.

The total flow time criterion is useful under certain circumstances, however this work focuses on problem of maximization of the probability, that the schedule is completed before single common due date. The proposed exact algorithms have the benefit that they do not produce symmetries during the search, as compared to [Ranjbar *et al.*, 2012].

1.2 Contribution

The contributions of this work are several:

(i) We propose an extension of the heuristic algorithm given by [Ranjbar *et al.*, 2012], which in many cases provides significantly better lower bound on the objective value.

(ii) We introduce a linear approximation of the non-linear model of the problem, which can be used as a standalone solver and as a method to solve a sub-problem of the RPMSP.

(iii) We provide a Branch-and-Price decomposition of the problem, which employs two support models (the Master Problem and Pricing Problem) and a branching scheme.

(iv) We propose two methods to solve the Pricing Problem.

(v) We describe an exact algorithm for a special case of two machines.

(vi) We conclude that our methods provide better results then the ones proposed by [Ranjbar *et al.*, 2012]

1.3 Thesis outline

In Section 2 we formally describe the problem and give bounds on the objective value. We also provide a non-linear model and introduce our proposed heuristic algorithm to solve the problem. In Section 3, we propose a Mixed-Integer Linear Program for the problem derived from the non-linear model. We also discuss bounds on the approximation intervals. The Branch-and-Price decomposition of the problem is introduced in Section 4. We give full description of how the decomposition was obtained and propose a way how to solve it. In Section 5 we show how to solve a special case of the problem, when there are exactly 2 machines present. Section 6 shows the results of experiments performed for each proposed method. We draw conclusions in Section 7.

2 Problem statement

Consider M identical parallel machines and a set of N jobs $\mathcal{J}$. We wish to distribute them among the machines such that a criterion function will be optimized. Let us denote j-th job as J_j. Each of these is given by a processing time π_j, which is not exactly known and is given as a normal probability distribution. In other words, we can associate processing time π_j of each job with some normal distribution, which is described by its mean μ_j and variance σ_j^2 :

$$\pi_j \sim \mathcal{N}(\mu_j, \sigma_j^2), \quad \mu_j, \sigma_j^2 \in \mathbb{N} \tag{2.1}$$

We also consider a common deadline δ, before which all the tasks should be completed. Our objective is to find a schedule that *maximizes the probability that we process all the tasks before the deadline* δ.

Let us consider a set of jobs $\mathcal{S}_i \subseteq \mathcal{J}$ to be scheduled on a machine M_i. Utilizing the property that the sum of independent normally distributed random variables has a normal distribution, the time for the machine to complete jobs $\mathcal{S}_i$ is given by a normal distribution $\mathcal{N}(\mu_{M_i}, \sigma_{M_i}^2)$, whose parameters are merely the sum of parameters of the jobs scheduled on this machine. In other words,

$$\mathcal{N}(\mu_{M_i}, \sigma_{M_i}^2) = \mathcal{N}\left(\sum_{j \in \mathcal{S}_i} \mu_j, \sum_{j \in \mathcal{S}_i} \sigma_j^2\right) \tag{2.2}$$

The probability, that machine M_i finishes processing its scheduled jobs before the deadline δ, is given by the cumulative standard normal distribution function Φ, as

$$\Pr[M_i \text{ finishes before } \delta] = \Phi\left(\frac{\delta - \mu_{M_i}}{\sqrt{\sigma_{M_i}^2}}\right). \tag{2.3}$$

Let us call the quantity defined in Equation (2.3) the *partial service level*. We denote the partial service level corresponding to i-th machine as Φ_i. Under the assumption of independence, the probability that *all* the machines finish processing before the deadline is a product of these probabilities, i.e.,

$$\Pr[\text{All jobs in } \mathcal{J} \text{ are completed before } \delta] = \prod_{i=1}^{M} \Phi\left(\frac{\delta - \mu_{M_i}}{\sqrt{\sigma_{M_i}^2}}\right). \tag{2.4}$$

We will denote this quantity as the *total service level*. In other words, the total service level is the product of all partial service levels. Since we want to maximize this probability, we state the objective as

$$\max \prod_{i=1}^{M} \Phi\left(\frac{\delta - \mu_{M_i}}{\sqrt{\sigma_{M_i}^2}}\right). \tag{2.5}$$

The problem defined in this way is known as a *robust parallel machine scheduling problem* [Ranjbar *et al.*, 2012]. In the Graham's three-field scheduling notation [Graham *et al.*, 1979], this problem can be denoted as $P|\pi_j \sim \mathcal{N}(\mu_j, \sigma_j^2)|\Pr[C_{\max} \leq \delta]$. This problem is known to be strongly $\mathcal{NP}$-hard [Ranjbar *et al.*, 2012].

As an example, consider an instance described by set of jobs shown in Table 1. Let $\delta = 84$.

J	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}
μ	32	12	2	20	19	35	23	21	26	41
σ^2	2	2	1	9	2	4	13	3	8	7

Table 1: Example of a problem instance.

Then, the optimal solution for the instance is

$$\begin{aligned} \{J_1, J_5, J_9\} &\mapsto M_1, \\ \{J_2, J_4, J_7, J_8\} &\mapsto M_2, \\ \{J_3, J_6, J_{10}\} &\mapsto M_3 \end{aligned}$$

and the service level is

$$\Phi\left(\frac{84-(32+19+26)}{\sqrt{2+2+8}}\right)\cdot\Phi\left(\frac{84-(12+20+23+21)}{\sqrt{2+9+13+3}}\right)\cdot\Phi\left(\frac{84-(2+35+41)}{\sqrt{1+4+7}}\right) = 0.8796\,. \tag{2.6}$$

Note that this is not the only optimal solution, because the job assignment can by done for any permutation of the machine set. We call such solutions *symmetrical*.

2.1 Non-linear model

With the objective function defined in the Equation (2.5), we can describe the problem with a non-linear mixed integer program. First, we introduce a binary assignment variables $x_{i,j}$, for which it hold that

$$x_{i,j} = \begin{cases} 1 & \text{if job } J_j \text{ is scheduled on machine } M_i \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

The sum of the means and variances of jobs scheduled on some machine M_i can be then written as

$$\mu_{M_i} = \sum_{j=1}^{N} x_{i,j} \cdot \mu_j \quad \forall i \in \{1, \ldots, M\}, \tag{2.8}$$

$$\sigma^2_{M_i} = \sum_{j=1}^{N} x_{i,j} \cdot \sigma_j^2 \quad \forall i \in \{1, \ldots, M\}. \tag{2.9}$$

The last thing left is to introduce a constraint that enforces that each job is scheduled exactly once on some machine. We summarize the model as:

$$\max \prod_{i=1}^{M} \Phi\left(\frac{\delta - \mu_{M_i}}{\sqrt{\sigma^2_{M_i}}}\right) \tag{2.10}$$

$$\text{subject to:} \quad \mu_{M_i} = \sum_{j=1}^{N} x_{i,j} \cdot \mu_j \quad \forall i \in \{1, \ldots, M\}, \tag{2.11}$$

$$\sigma^2_{M_i} = \sum_{j=1}^{N} x_{i,j} \cdot \sigma^2_j \quad \forall i \in \{1, \ldots, M\}, \tag{2.12}$$

$$\sum_{i=1}^{M} x_{i,j} = 1 \quad \forall j \in \{1, \ldots, N\} \tag{2.13}$$

$$\text{where:} \quad x_{i,j} \in \{0, 1\}, \quad \forall i \in \{1, \ldots, M\}, \forall j \in \{1, \ldots, N\} \tag{2.14}$$

The Equation (2.13) represents the constraint that forces the scheduling of each job exactly once. To solve this problem, one can use an off-the-shelf non-linear mixed-integer program solver with model (2.10)–(2.13). However, as it will be further shown, its performance is rather poor. Hence, one needs to develop more efficient approaches.

2.2 Problem properties

In this section, we discuss the properties of the problem and give lower and upper bounds on the optimal objective value that can be calculated for a given problem instance. We also introduce our own heuristic algorithm for the problem.

In their work, [Ranjbar *et al.*, 2012] introduce a method to calculate the lower and upper bounds on a problem instance. The calculation of the lower bound is given by their proposed initial heuristic and the upper bound is provided by considering a best-case scenario by scheduling what their denote as virtual jobs. Both these bounds can then be updated during the run of their proposed algorithms to solve a problem instance. Furthermore, they introduce a lower bound on the number of jobs that each machine has to schedule, denoted as v_{min}. This bound is initially set to 1 and is updated every time the upper bound is recalculated.

We now discuss several properties of the problem. Consider a case where it holds that $N \leq M$ (i.e the number of machines is larger then or equal to the number of jobs to schedule). For such problem, the solution is straightforward since we can schedule one job per machine, potentially leaving some of them empty. For the cases, where it holds that $N > M$, each machine has to schedule at least one job, as reflected in the initial value of the v_{min} bound. This can be shown from the following example. Consider an instance with M machines and N jobs to schedule, where $N > M$. Let the optimal objective value by given as

$$F_{\text{opt}_M} = \prod_{i=1}^{M} \Phi\left(\frac{\delta - \mu_{M_i}}{\sqrt{\sigma^2_{M_i}}}\right). \tag{2.15}$$

Now consider an instance with $M+1$ machines and the same set of N jobs and optimal objective value $F_{\text{opt}_{M+1}}$, where $N \geq M+1$. Since this is a *relaxation* of the same instance with M machines, it must hold that

$$F_{\text{opt}_{M+1}} \geq F_{\text{opt}_M}. \tag{2.16}$$

To obtain the same objective value, we simply use the same schedule as in the M machine version of the problem and leave some single machine empty, so the objective value F_{obj} is given as

$$F_{\text{obj}} = \lim_{\sigma^2_{M_i} \to 0} F_{\text{opt}_M} \cdot \Phi\left(\frac{\delta - 0}{\sqrt{\sigma^2_{M_i}}}\right) = F_{\text{opt}_M}. \tag{2.17}$$

Therefore, to obtain a better objective value, at least one job must be scheduled on each machine.

2.2.1 v_{max} calculation

In this section, we prove an upper bound on the number of jobs that each machine can process in an optimal schedule. We will denote this bound as v_{max}.

Proposition 1. *Let $\mathcal{J}^*$ be an ordered set of virtual jobs described by two parameters $(\mu_{[j]}, \sigma^2_{[j]})$. Each such virtual job is created by sorting the parameters of the original jobs $J \in \mathcal{J}$ for a given instance in an ascending order, so that it holds*

$$\mu_{[j]} \leq \mu_{[j+1]}, \quad \forall j \in \{1, ..., N-1\} \tag{2.18}$$

$$\sigma^2_{[j]} \leq \sigma^2_{[j+1]}, \quad \forall j \in \{1, ..., N-1\}. \tag{2.19}$$

Let O_{LB} be some lower bound on the objective value of a problem instance. Then the upper bound on the maximum number of jobs each machine can schedule v_{max} is such a number for which it holds that

$$\Phi\left(\frac{\delta - \sum_{j=1}^{v_{max}} \mu_{[j]}}{\sum_{j=1}^{v_{max}} \sigma^2_{[j]}}\right) \geq O_{LB} \tag{2.20}$$

$$\Phi\left(\frac{\delta - \sum_{j=1}^{\min(v_{max}+1, N)} \mu_{[j]}}{\sum_{j=1}^{\min(v_{max}+1, N)} \sigma^2_{[j]}}\right) \leq O_{LB}, \tag{2.21}$$

where N is the number of jobs for given instance.

Proof. Assume that we schedule first k virtual jobs from the set $\mathcal{J}^*$ to some single machine in a consecutive order. Let $\mathcal{S} \subseteq \mathcal{J} : |\mathcal{S}| \geq k$. It then must hold that

$$\sum_{j=1}^{k} \mu_{[j]} \leq \sum_{j=1}^{k} \mu_j, \tag{2.22}$$

$$\sum_{j=1}^{k} \sigma^2_{[j]} \leq \sum_{j=1}^{k} \sigma^2_j, \tag{2.23}$$

where $[\mu_j, \sigma_j^2] \sim J_j \in \mathcal{S}$. It immediately follows that

$$\Phi\left(\frac{\delta - \sum_{j=1}^{k} \mu_{[j]}}{\sqrt{\sum_{j=1}^{k} \sigma^2_{[j]}}}\right) \geq \Phi\left(\frac{\delta - \sum_{j=1}^{k} \mu_j}{\sqrt{\sum_{j=1}^{k} \sigma^2_j}}\right). \tag{2.24}$$

Therefore, assigning k first jobs from the ordered set $\mathcal{J}^*$ can never give worse objective value than assigning any k jobs from the set of the original jobs $\mathcal{J}$. Next, note that for any partial service level Φ_i it holds that

$$\Phi_i \geq \prod_{l=1}^{M} \Phi_l \tag{2.25}$$

since

$$\Phi_i \in [0, 1], \quad \forall i \in \{1, \ldots, M\}, \tag{2.26}$$

where M is the number of machines for the given instance. This implies that if a single partial service level drops below the lower bound O_{LB}, then the whole objective value can never be higher then this lower bound. Then from the Equation (2.24) it immediately follows

$$\Phi\left(\frac{\delta - \sum_{j=1}^{k} \mu_{[j]}}{\sqrt{\sum_{j=1}^{k} \sigma^2_{[j]}}}\right) \geq \Phi\left(\frac{\delta - \sum_{j=1}^{k} \mu_j}{\sqrt{\sum_{j=1}^{k} \sigma^2_j}}\right) \geq O_{LB}. \tag{2.27}$$

The upper bound on the maximum number of jobs each machine can schedule v_{max} is then such a number k, for which this equation holds, but does not for $k = v_{max} + 1$. Furthermore, we note that v_{max} bound is computable in $\mathcal{O}(n \log n)$ time given the value O_{LB}. □

2.2.2 Large Job Allocated First heuristic

In this section, we propose an extension of the heuristic algorithm given by [Ranjbar *et al.*, 2012]. Their method produces a feasible solution to the problem whose objective value is then taken as the lower bound for the instance. However, they do not do any sorting of the jobs beforehand and schedule them in a unsorted order. Our improvement is to introduce an ordering that would simulate the scheduling of "large" jobs first (i.e., jobs that would most likely take a long time to process). We denote this method as *Large Job Allocated First* or *LJAF* heuristic.

Consider an instance described by M machines and N jobs. The proposed heuristic is

shown in Algorithm 1. It sorts jobs in a non-increasing order of products of means and variances $\mu_j \cdot \sigma_j^2$. The ties are broken by larger μ_j. In each step, a job is taken and assigned to the machine $i \in \{1, \ldots, M\}$ with the currently largest probability π_i.

Algorithm 1 LJAF heuristics

let $\mu_1 \cdot \sigma_1^2 \geq \mu_2 \cdot \sigma_2^2 \geq \ldots \geq \mu_N \cdot \sigma_N^2$
$c_i \leftarrow 1 \quad \forall i \in \{1, \ldots, M\}$
$\boldsymbol{a}_i \leftarrow \mathbf{0} \quad \forall i \in \{1, \ldots, M\}$
for $j \leftarrow 1$ **to** N **do**
  $i^\star \leftarrow \arg\max_{i \in \{1,\ldots,M\}} c_i$
  $a_{i^\star,j} \leftarrow 1$
  $c_i \leftarrow \Phi\left(\frac{\delta - \boldsymbol{\mu}^T \boldsymbol{a}_{i^\star}}{\sqrt{\boldsymbol{\sigma}^T \boldsymbol{a}_{i^\star}}}\right)$
end for
return $c_i, \boldsymbol{a}_i \quad \forall i \in \{1, \ldots, M\}$

This algorithm is analogical to the LPT-rule (longest processing time first) for $P||C_{max}$ problem.

3 Mixed-Integer Linear Program model

In this section, we develop a mixed-integer linear program model to solve the problem. All the non-linear operations are present in the objective function, namely the product, the cumulative distribution function, the division and the square root.

We introduce a model transformation that allows an easier linear approximation. Then, we proceed with the actual piece-wise linear model of the non-linear objective function. Moreover, we establish finite bounds on the intervals where the non-linearities need to be approximated.

3.1 Model transformation

We begin by transforming the objective function given by Equation (2.10). First, let us replace the product by exploiting the logarithm property. Hence, we have

$$\max \ln \left(\prod_{i=1}^{M} \Phi \left(\frac{\delta - \mu_{M_i}}{\sqrt{\sigma_{M_i}^2}} \right) \right) = \max \sum_{i=1}^{M} \ln \Phi \left(\frac{\delta - \mu_{M_i}}{\sqrt{\sigma_{M_i}^2}} \right), \tag{3.1}$$

where $\ln(x)$ represents the natural logarithm. In the next step, we change the objective function by using a substitution $\sigma_{M_i} \equiv \sqrt{\sigma_{M_i}^2}$. The right-hand side of the equation (3.1) can then be rewritten as

$$\max \sum_{i=1}^{M} \ln \Phi \left(\frac{\delta - \mu_{M_i}}{\sigma_{M_i}} \right). \tag{3.2}$$

Now we show how to linearize the division of two continuous variables. We use a similar trick that is used in the linear fractional programming, known as *Charnes-Cooper transformation* [Charnes and Cooper, 1962]. By introducing a variable $t_i \equiv \frac{1}{\sigma_{M_i}}$ and by using the substitution $v_i \equiv t_i \cdot \mu_{M_i}$, we transform the objective function into

$$\max \sum_{i=1}^{M} \ln \Phi(\delta \cdot t_i - v_i). \tag{3.3}$$

We have obtained a new transformed objective function, where the only non-linear term is the logarithm of the cumulative distribution function. The transformation of the objective function has introduced several new variables and these have to be reflected in the constraints as well. We begin by rewriting the original constraints first. Multiplying the equation (2.11) with t_i, we get:

$$v_i = \mu_{M_i} \cdot t_i = \sum_{j=1}^{N} x_{i,j} \cdot t_i \cdot \mu_j \quad \forall i \in \{1, \ldots, M\} \tag{3.4}$$

We set $z_{i,j} \equiv x_{i,j} \cdot t_i$ to get the first transformed constraint

$$v_i = \sum_{j=1}^{N} z_{i,j} \cdot \mu_j \quad \forall i \in \{1, \ldots, M\}. \tag{3.5}$$

Note that the definition of $x_{i,j}$ and t_i necessarily implies that $z_{i,j} \in \{0, t_i\}$. Similarly, we transform the Equation (2.12) by multiplying both sides with t_i:

$$\sigma^2_{M_i} \cdot t_i = \sum_{j=1}^{N} x_{i,j} \cdot t_i \cdot \sigma_j^2 \quad \mapsto \quad u_i = \sum_{j=1}^{N} z_{i,j} \cdot \sigma_j^2 \quad \forall i \in \{1, \ldots, M\}, \tag{3.6}$$

where we let $u_i \equiv \sigma^2_{M_i} \cdot t_i$.

Finally, we transform Equation (2.13). However, due to the fact that t_i is tied to specific machine and the constraint sums across all the machines, we cannot simply multiply both sides by some single t_i. Instead, we need to employ the following trick. We introduce a new binary variable $w_{i,j}$ which will serve the same purpose as variable $x_{i,j}$ in the original model (i.e. the job-machine assignment). We write that

$$\sum_{i=1}^{M} w_{i,j} = 1 \quad \forall j \in \{1, \ldots, N\} \tag{3.7}$$

to enforce that each job is scheduled exactly once. Now we need to create a relation between the variables $w_{i,j}$ and $z_{i,j}$. Whenever $w_{i,j}$ is set to 1, the corresponding $z_{i,j}$ must be equal to its corresponding t_i. When we introduce some large constant $\mathbf{M}$, such that $\mathbf{M} \to \infty$, we can add the following constraints:

$$t_i - (1 - w_{i,j})\mathbf{M} \leq z_{i,j} \quad \forall i \in \{1, \ldots, M\}, \forall j \in \{1, \ldots, N\} \tag{3.8}$$

$$t_i + (1 - w_{i,j})\mathbf{M} \geq z_{i,j} \quad \forall i \in \{1, \ldots, M\}, \forall j \in \{1, \ldots, N\} \tag{3.9}$$

$$w_{i,j} \geq z_{i,j}\mathbf{M} \quad \forall i \in \{1, \ldots, M\}, \forall j \in \{1, \ldots, N\} \tag{3.10}$$

Indeed, if some $w_{i,j}$ is equal to 1, it will force the corresponding $z_{i,j}$ to take on the given t_i value. On the other hand, if $w_{i,j}$ will amount to zero, $z_{i,j}$ has to be zero as well. Note that in Equation (3.6) we have introduced a new variable u_i. From the definition of t_i, we set

$$\sigma^2_{M_i} = \frac{1}{t_i^2}. \tag{3.11}$$

However, from the definition of u_i we get that

$$u_i = \sigma^2_{M_i} t_i = \frac{t_i}{t_i^2}. \tag{3.12}$$

Therefore,

$$u_i = \frac{1}{t_i}. \tag{3.13}$$

The transformed non-linear model is given as follows:

$$\max \sum_{i=1}^{M} \ln \Phi(\delta \cdot t_i - v_i) \tag{3.14}$$

$$\text{subject to:} \quad v_i = \sum_{j=1}^{N} z_{i,j} \cdot \mu_j \quad \forall i \in \{1, \ldots, M\}, \tag{3.15}$$

$$u_i = \sum_{j=1}^{N} z_{i,j} \cdot \sigma_j \quad \forall i \in \{1, \ldots, M\}, \tag{3.16}$$

$$\sum_{i=1}^{M} w_{i,j} = 1 \quad \forall j \in \{1, \ldots, N\}, \tag{3.17}$$

$$t_i - (1 - w_{i,j})\mathbf{M} \leq z_{i,j} \quad \forall i \in \{1, \ldots, M\}, \forall j \in \{1, \ldots, N\}, \tag{3.18}$$

$$t_i + (1 - w_{i,j})\mathbf{M} \geq z_{i,j} \quad \forall i \in \{1, \ldots, M\}, \forall j \in \{1, \ldots, N\}, \tag{3.19}$$

$$w_{i,j} \geq z_{i,j}\mathbf{M} \quad \forall i \in \{1, \ldots, M\}, \forall j \in \{1, \ldots, N\}, \tag{3.20}$$

$$u_i = \frac{1}{t_i} \quad \forall i \in \{1, \ldots, M\} \tag{3.21}$$

where:

$$t_i, v_i, u_i \in \mathbb{R}^+, \quad \forall i \in \{1, \ldots, M\}, \tag{3.22}$$

$$z_{i,j} \in \mathbb{R}^+, \quad \forall i \in \{1, \ldots, M\}, \forall j \in \{1, \ldots, N\}, \tag{3.23}$$

$$w_{i,j} \in \{0, 1\}, \quad \forall i \in \{1, \ldots, M\}, \forall j \in \{1, \ldots, N\} \tag{3.24}$$

3.2 Linear approximation

We are now left with two non-linear expressions to deal with – the fraction (3.21) and the natural logarithm of the cumulative normalized normal distribution function (3.14). To get rid of the fraction first, the strategy will be to approximate it around several points across the whole interval where we need to perform the linearization. This will naturally divide the interval into L sub-intervals, where we approximate the function. Depending on the argument in the fraction, we choose such sub-interval, which provides the best approximation. We approximate the function $\frac{1}{t_i}$ using the Taylor expansion. We write down the general formula up to the first order as

$$T(f(x), a) = f(a) + \frac{df}{dx}(a) \cdot (x - a) + \varepsilon, \tag{3.25}$$

where ε is the error term. Applying to our case, we get

$$T\left(\frac{1}{t_i}, h_k\right) \approx \frac{1}{h_k} - \frac{1}{h_k^2}(t_i - h_k), \tag{3.26}$$

with h_k representing the values around which we perform the approximation. These points represent the centers of the above mentioned sub-intervals. Note that the length of two arbitrary sub-intervals does not have to be equal.

Now we introduce the corresponding constraints for the model that produce the necessary effect. The way we go about it is following. We introduce a new variable $\lambda_{i,k}$, which is equal to the value of the linearization performed on k-th sub-interval, for the i-th machine. Further, we enforce that if the fraction argument on the i-th machine falls into the k-th sub-interval, only the corresponding $\lambda_{i,k}$ value evaluates to its corresponding linearization, while the other $\lambda_{i,k}$ values for the same machine are set to zero. This means that we can write u_i as:

$$u_i = \sum_{k=1}^{L} \lambda_{i,k} \quad \forall i \in \{1, \ldots, M\}. \tag{3.27}$$

Figure 1 shows an example piece-wise approximation of the division on interval $[0.05, 0.3]$. The yellow dotted lines indicate division into the sub-intervals. The first order Taylor expansion is evaluated around the center of each of the sub-intervals. The values shown on the x-axis represent the chosen h_k values, each having a corresponding $\lambda_{i,k}$ value, where the i-th value is given by the machine on which we perform the approximation. If the corresponding t_i value falls into the sub-interval centered around some point h_j, only $\lambda_{i,j}$ value is evaluated using the corresponding linear approximation. Every other $\lambda_{i,k}$ for the given machine is set to zero.

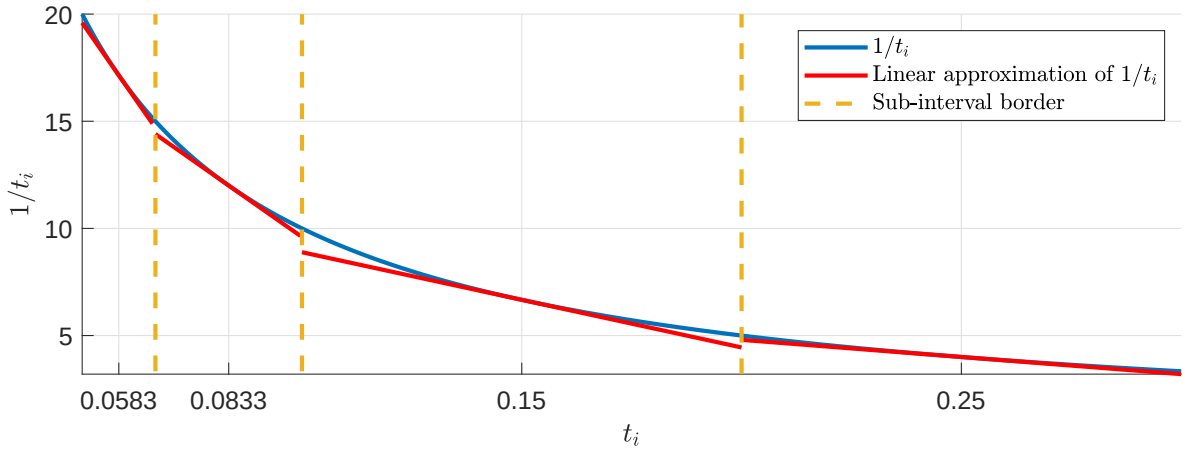


Figure 1: An example of piece-wise linear approximation of the fraction.

As mentioned above, when we fix the i index, the u_i will be equal only to one specific value $\lambda_{i,k}$ since others will be zero. To be able to do this though, we need to introduce another binary decision variable $y_{i,k}$ for which it will hold that

$$y_{i,k} = \begin{cases} 1 & \text{if the corresponding } t_i \text{ value falls into the } k\text{-th subinterval} \\ 0 & \text{otherwise.} \end{cases} \tag{3.28}$$

We now have everything we need to finish the linearization. Let Δh_k be the length of the k-th sub-interval. In other words, k-th sub-interval can be written as $[h_k - \frac{\Delta h_k}{2}, h_k + \frac{\Delta h_k}{2}]$. Remember, we want to linearize the term $\frac{1}{t_i}$, so the t_i is our argument. If t_i falls into some sub-interval given by the point h_k, it means that

$$h_k - \frac{\Delta h_k}{2} \leq t_i \leq h_k + \frac{\Delta h_k}{2}. \tag{3.29}$$

By splitting this into two inequalities and a subtraction we get the following:

$$-\frac{\Delta h_k}{2} \leq t_i - h_k \tag{3.30}$$

$$t_i - h_k \leq \frac{\Delta h_k}{2} \tag{3.31}$$

The only thing left to do is to incorporate the $y_{i,k}$ variable to these inequalities so they switch the corresponding assignment variables accordingly. We introduce the following constraints:

$$t_i - h_k \geq -(1 - y_{ik})\boldsymbol{M} - \frac{\Delta h_k}{2} \quad \forall i \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, L\}, \tag{3.32}$$

$$t_i - h_k \leq \frac{\Delta h_k}{2} + (1 - y_{ik})\boldsymbol{M} \quad \forall i \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, L\} \tag{3.33}$$

We see that at least one set of equations (3.32)–(3.33) has to hold for the $y_{i,k}$ variables to be switched. It is also desired that for each machine only one $y_{i,k}$ is activated, so we introduce additional constraint

$$\sum_{k=1}^{L} y_{i,k} = 1 \quad \forall i \in \{1, \ldots, M\}. \tag{3.34}$$

Finally, with these constraints set, we can assign the corresponding linear approximations

$$\lambda_{i,k} \geq \frac{1}{h_k} - \frac{1}{h_k^2}(t_i - h_k) - (1 - y_{ik})\boldsymbol{M} \quad \forall i \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, L\}, \tag{3.35}$$

$$\lambda_{i,k} \leq \frac{1}{h_k} - \frac{1}{h_k^2}(t_i - h_k) + (1 - y_{ik})\boldsymbol{M} \quad \forall i \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, L\}, \tag{3.36}$$

$$\lambda_{i,k} \leq y_{i,k}\boldsymbol{M} \quad \forall i \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, L\}. \tag{3.37}$$

These constraints force the $\lambda_{i,k}$ to evaluate to its corresponding value, or to zero if we are not evaluating the approximation on the given sub-interval.

3.3 Complete model

Only non-linear term left is the objective function with its cumulative distribution function. We could proceed similarly as in the fraction linearization, however we choose no to due to the following:

(i) Cumulative normal distribution function does not have an analytic form

(ii) In our implementation we use solver feature that has support for *piece-wise linear approximation of the objective function* implemented by a specialized simplex method.

The complete model can be stated as follows:

$$\max \sum_{i=1}^{M} \ln \Phi(\delta \cdot t_i - v_i) \tag{3.38}$$

$$\text{subject to:} \quad v_i = \sum_{j=1}^{N} \mu_j \cdot z_{i,j} \quad \forall i \in \{1, \ldots, M\}, \tag{3.39}$$

$$u_i = \sum_{j=1}^{N} \sigma_j^2 \cdot z_{i,j} \quad \forall i \in \{1, \ldots, M\}, \tag{3.40}$$

$$\sum_{i=1}^{M} w_{i,j} = 1 \quad \forall j \in \{1, \ldots, N\}, \tag{3.41}$$

$$t_i - (1 - w_{i,j})\boldsymbol{M} \leq z_{i,j} \quad \forall i \in \{1, \ldots, M\}, \forall j \in \{1, \ldots, N\}, \tag{3.42}$$

$$z_{i,j} \leq t_i + (1 - w_{i,j})\boldsymbol{M} \quad \forall i \in \{1, \ldots, M\}, \forall j \in \{1, \ldots, N\}, \tag{3.43}$$

$$z_{i,j} \leq w_{i,j}\boldsymbol{M} \quad \forall i \in \{1, \ldots, M\}, \forall j \in \{1, \ldots, N\}, \tag{3.44}$$

$$\sum_{j=1}^{N} z_{i,j} \geq t_i \quad \forall i \in \{1, \ldots, M\}, \tag{3.45}$$

$$u_i = \sum_{k=1}^{L} \lambda_{i,k} \quad \forall i \in \{1, \ldots, M\}, \tag{3.46}$$

$$\sum_{k=1}^{L} y_{i,k} = 1 \quad \forall i \in \{1, \ldots, M\}, \tag{3.47}$$

$$\lambda_{i,k} \geq \frac{1}{h_k} - \frac{1}{h_k^2}(t_i - h_k) - (1 - y_{i,k})\boldsymbol{M} \quad \forall i \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, L\}, \tag{3.48}$$

$$\lambda_{i,k} \leq \frac{1}{h_k} - \frac{1}{h_k^2}(t_i - h_k) + (1 - y_{i,k})\boldsymbol{M} \quad \forall i \in \{1, \ldots, M\} \forall k \in \{1, \ldots, L\}, \tag{3.49}$$

$$\lambda_{i,k} \leq y_{i,k}\boldsymbol{M} \quad \forall i \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, L\}, \tag{3.50}$$

$$t_i - h_k \geq -(1 - y_{i,k})\boldsymbol{M} - \frac{\Delta h_k}{2} \quad \forall i \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, L\}, \tag{3.51}$$

$$t_i - h_k \leq \frac{\Delta h_k}{2} + (1 - y_{i,k})\boldsymbol{M} \quad \forall i \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, L\} \tag{3.52}$$

where:

$$t_i, v_i, u_i \in \mathbb{R}^+ \quad \forall i \in \{1, \ldots, M\}, \tag{3.53}$$

$$z_{i,j} \in \mathbb{R}^+ \quad \forall i \in \{1, \ldots, M\}, \forall j \in \{1, \ldots, N\}, \tag{3.54}$$

$$\lambda_{i,k} \in \mathbb{R}^+ \quad \forall i \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, L\}, \tag{3.55}$$

$$w_{i,j} \in \{0, 1\} \quad \forall i \in \{1, \ldots, M\}, \forall j \in \{1, \ldots, N\}, \tag{3.56}$$

$$y_{i,k} \in \{0, 1\} \quad \forall i \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, L\} \tag{3.57}$$

3.4 Model tuning

From the practical standpoint of view we also need to consider an additional thing and that is the actual implementation of the approximation (sub)intervals. It is obvious that the smoother the interval division is, the more precise results will be. On the other hand, smooth division induces many sub-intervals and therefore causes longer running time of the solver. Therefore, we impose bounds on the approximation intervals to make them as narrow as possible. Note that in theory, it can be done precisely, since our instances work only with integers. We then try to find a trade-off between the precision of the model and the smoothness of the approximation.

3.4.1 Fraction approximation interval tuning

In this section, we establish bounds on the interval on which we will approximate the function $u_i = \frac{1}{t_i}$, by using the v_{min} and v_{max} bounds described in Section 2. We also introduce a method in which the sub-intervals are generated. We begin by discussing the lower bound. We wish to minimize the value of u_i which in turn means maximizing the value of t_i. Remember that this value is defined as $t_i = \frac{1}{\sigma_{M_i}}$, $\sigma_{M_i} = \sqrt{\sigma^2_{M_i}}$. Therefore, we want to make the value of σ_{M_i} as small as possible. The first option would be to take the lowest possible value of σ^2_j which corresponds to scheduling a single job. We know, that each machine has to schedule at least v_{min} jobs. Let $\boldsymbol{\sigma}^*$ denote an ordered set of all available variances. Let $\sigma^2_{[i]}$ denote an i-th element of the set $\boldsymbol{\sigma}^*$. It holds that

$$\sigma^2_{[j]} \leq \sigma^2_{[j+1]}, \quad \forall j \in \{1, \ldots, N-1\}. \tag{3.58}$$

The lower bound F_{LB} of the fraction approximation interval is then given as

$$F_{LB} = \frac{1}{t_{max}} = \sqrt{\sum_{j=1}^{v_{min}} \sigma^2_{[j]}}. \tag{3.59}$$

To get the upper bound we need to make the term u_i as large as possible. This implies scheduling as many jobs with as high variances as possible. Since we know that no machine can schedule more than v_{max} jobs, the upper bound F_{UB} is given as

$$F_{UB} = \frac{1}{t_{min}} = \sqrt{\sum_{j=N-v_{max}+1}^{N} \sigma^2_{[j]}}. \tag{3.60}$$

We now turn attention to the question as how to design the sub-intervals. There are many possible approaches, the most obvious being creating equidistant splits of the whole domain of the interval. However, we decided to split the interval equidistantly with respect to the range of the approximated function. Let b_k, b_{k+1} be the two boundary points of k-th sub-interval. We require that

$$\left| \frac{1}{b_k} - \frac{1}{b_{k+1}} \right| = \text{const.} \equiv \Delta H, \quad \forall k \in \{1, \ldots, L\}. \tag{3.61}$$

Since $\frac{1}{t_i}$ is monotonously decreasing function, we omit the absolute value. From the equation above we derive a recursive formula for the calculation of the boundary points that represent the sub-intervals:

$$\frac{1}{b_k} - \frac{1}{b_{k+1}} = \Delta H \tag{3.62}$$

$$\frac{b_{k+1}}{b_k} - 1 = \Delta H \cdot b_{k+1} \tag{3.63}$$

$$b_{k+1}\left(\frac{1}{b_k} - \Delta H\right) = 1 \tag{3.64}$$

$$b_{k+1} = \frac{b_k}{1 - \Delta H \cdot b_k} \tag{3.65}$$

It holds that $b_1 = F_{LB}$, $b_k \in [F_{LB}, F_{UB}], \forall k \in \{2, \ldots, L\}$ and $b_{L+1} = F_{UB}$. The central point h_k of the $k-th$ sub-interval is then given as

$$h_k = \frac{b_k + b_{k+1}}{2}. \tag{3.66}$$

3.4.2 Objective function approximation interval tuning

This section describes lower and upper bounds placed on the objective function. To establish the lower bound, we calculate a heuristic solution to obtain some feasible schedule. The lower bound O_{LB} is then given as the logarithm of the service level calculated from the heuristic solution.

To obtain the upper bound, we need to consider the best case scenario. Let $\mathcal{J}^*$ denote an ordered set of virtual jobs created from the set of the original jobs $\mathcal{J}$, by recombining the means and variances of the jobs in such way that it holds that $\mu_{[j]} \leq \mu_{j+1}$ and $\sigma^2_{[j]} \leq \sigma^2_{j+1}$ for every job $J_{[j]} \in \mathcal{J}^*$. The upper bound O_{UB} on the objective value is then given as

$$O_{UB} = \ln \Phi \left(\frac{\delta - \sum_{j=1}^{v_{min}} \mu_{[j]}}{\sqrt{\sum_{j=1}^{v_{min}} \sigma^2_{[j]}}} \right). \tag{3.67}$$

To create the division of the interval into R sub-intervals we use and equidistant split in the domain of the function, so for the two boundary points o_k, o_{k+1} of the k-th sub-interval, it holds that

$$|o_k - o_{k+1}| = \Delta F, \quad \forall k \in \{1, \ldots, R\}. \tag{3.68}$$

Since the objective function is increasing, we can again omit the absolute value but we need to change the sign. The recursive relation for the sub-interval boundaries is then given as

$$o_{k+1} = \Delta F + o_k \tag{3.69}$$

It holds that $o_1 = \Phi^{-1}(O_{LB})$, $o_k \in [\Phi^{-1}(O_{LB}), \Phi^{-1}(O_{UB})], \forall k \in \{2, \ldots, R\}$ and $o_{R+1} = \Phi^{-1}(O_{UB})$, where $\Phi^{-1}(x)$ is the inverse cumulative normal distribution function known as the *probit function*. We do not calculate the central points, since the approximation is done automatically by the Gurobi solver piece-wise linear objective feature, from the interval boundary points.

3.4.3 M constant tuning

In this section, we discuss how to set properly the $\boldsymbol{M}$ constant. Since the big $\boldsymbol{M}$ constant arises in various sets of constraints, we are going to make the following distinction:

$$\boldsymbol{M} \mapsto \boldsymbol{M}_1 \quad \text{for constraints (3.42) – (3.44)} \tag{3.70}$$

$$\boldsymbol{M} \mapsto \boldsymbol{M}_2 \quad \text{for constraints (3.48) – (3.50)} \tag{3.71}$$

$$\boldsymbol{M} \mapsto \boldsymbol{M}_3 \quad \text{for constraints (3.51) – (3.52)} \tag{3.72}$$

The constant $\boldsymbol{M}_1$ sets the $z_{i,j}$ variables. If we consider the worst case scenario ($t_i = t_{\max}, w_{i,j} = 0$), we see that by letting $\boldsymbol{M}_1 = t_{\max} = \frac{1}{\min(\boldsymbol{\sigma})}$ the constraints will still set the variables properly:

$$\begin{aligned} t_{\max} - t_{\max} &\leq z_{i,j} \\ z_{i,j} &\leq t_{\max} + t_{\max} \\ z_{i,j} &\leq 0. \end{aligned}$$

The constant $\boldsymbol{M}_2$ sets the $\lambda_{i,k}$ variables. We consider the worst case scenario and set $\boldsymbol{M}_2 = \lambda_{\max} = \frac{1}{F_{LB}}$. Note that from Figure 1 we see that the actual approximation lies beneath the actual function. We could therefore provide even tighter bound by evaluating the approximation at the lower bound as

$$\boldsymbol{M}_2 = \frac{1}{h_1} - \frac{1}{h_1^2}(F_{LB} - h_1). \tag{3.73}$$

Finally $\boldsymbol{M}_3$ variable arises in constraint that are responsible for setting the proper subinterval switching variable $y_{i,k}$. We set $\boldsymbol{M}_3 = F_{UB}$. In the worst scenario it holds that $t_i = F_{UB}, h_k = h_1$ and $y_i, 1 = 0$. We see that

$$\begin{aligned} F_{UB} - h_1 &\geq -F_{UB} - \frac{\Delta h_1}{2}, \\ F_{UB} - h_1 &\leq \frac{\Delta h_1}{2} + F_{UB}. \end{aligned}$$

4 Branch-and-Price model

In this chapter, we propose a Branch-and-Price [Barnhart *et al.*, 1998] decomposition to solve the problem. In general, Branch-and-Price algorithms consist of two support models: the Master Problem and the Pricing Problem. The Master Problem is a reformulation of the original problem, so that we don't assign the jobs one by one, but instead create sets of jobs (denoted as *patterns*) which we assign to machines. Let M denote the number of machines for given instance. We then select M patterns such that they maximize the objective function, given the constraint that each job is scheduled.

There is an exponential number of possible patterns, so we start by assuming only a small portion of them. This is called a *restricted Master Problem*. Depending on the solution of this problem, we generate additional patterns by a procedure known as the *column generation* [Desaulniers *et al.*, 2006].

In the column generation procedure, we pass the dual variables from the Master Problem, which represent costs of scheduling specific jobs, to the Pricing Problem, which in turn generates additional pattern. The Pricing Problem is directly derived from the dual formulation of the Master Problem. We update the already existing set of patterns with its solution and solve the Master Problem again. We repeat the process until we cannot find a pattern that has a negative *reduced cost* (i.e., the objective value of the Pricing Problem).

The Master Problem relaxes the pattern assignment variables and may not provide an integer solution. In that case, we need to apply a branching scheme, that creates two copies of the original problem, but imposes a constraint that some two jobs must be scheduled together or a constraint that the same jobs cannot be scheduled together. We then attempt to solve both those problems. If some of them does not yield an integer solution, we must perform the branching decision again. This creates a branching tree.

Once all the leaves of the branching tree are closed (i.e., integer solution was found) we take the best solution, which then represents the optimal solution to the given instance.

The whole procedure is shown in Figure 2. The initial heuristic generates the original set of patterns, which are then used in the Master Problem. The dual costs from the Master Problem are used in the Pricing Problem to get a new pattern. The Master problem is solved again with the updated set of patterns and recalculates the dual costs. The procedure repeats itself until no pattern with negative reduced cost can be found anymore. If the solver does not yield and integer solution we apply the branching mechanism. Once all the leaves of the branching tree are closed, we pick the best solution.

The main advantage of the Branch-and-Price approach is that we get rid of the symmetries while searching for the solution and that we push the non-linearity to separate procedure. This is a consequence of omitting the job-machine assignment and replacing it with the pattern-machine assignment method. In the following sections, we describe each part of the algorithm in details.

4.1 Master Problem

In the original non-linear model of the problem, we assigned individual jobs to machines. In Branch-and-Price decomposition, we use the variable lifting procedure. We create the set of all possible combinations of jobs on a single machine. Then, one could choose exactly

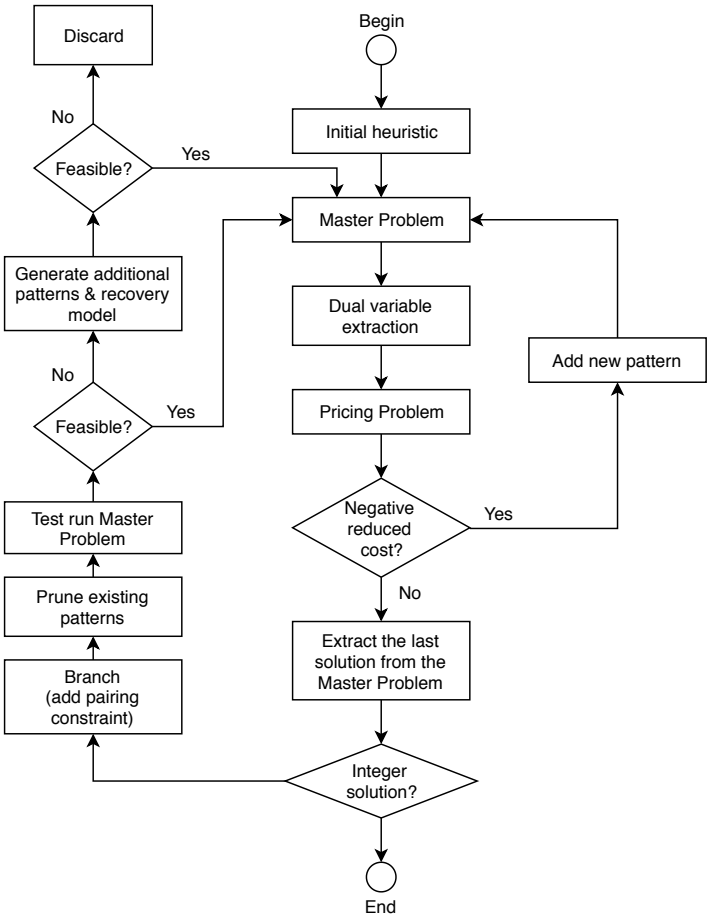


Figure 2: Diagram of the proposed Branch-and-Price algorithm.

M (i.e., the number of machines) of such combinations to form a feasible schedule, so that the objective function is maximized subject to condition that each job is scheduled. This is the foundation of Master Problem definition. We represent each such combination as a binary vector $\boldsymbol{p}_k = \{0,1\}^N$, where N is the number of jobs to schedule. We call such vector a *pattern*. As an example, consider an instance of 4 jobs and 2 machines. An example pattern might be $\boldsymbol{p}_s = (1,1,0,0)$. This indicates that the pattern schedules jobs 1 and 2. When we say that we assign (schedule) this pattern to machine 1, we mean that we schedule jobs 1 and 2 on the machine numbered as 1.

It immediately follows that to cover all the possibilities of job-machine assignments, we would have to generate 2^N patterns. Let us denote the set of all patterns by $\mathcal{P}$. However, complete representation of $\mathcal{P}$ is costly. The trick of the Branch-and-Price approach is that we can start with some initial subset of patterns, which can be relatively small and we can generate further suitable patterns along the way. We denote this reduced set of patterns as $\mathcal{P}'$. Moreover, in order to prove optimality of the solution, one does not typically need to generate the whole set $\mathcal{P}$.

Additionally, we define a cost of k-th pattern, denoted as $\ln(c_k)$, which is calculated as the logarithm of the partial service level of a single machine on which we schedule the given pattern.

As an example consider again the pattern $\boldsymbol{p}_s = (1, 1, 0, 0)$. Lets say that the instance is given by deadline $\delta = 35$, two parallel machines $M = 2$ and jobs $J_1 = (12, 7), J_2 = (18, 2), J_3 = (7, 3), J_4 = (13, 5)$. The cost $\ln(c_s)$ of pattern $\boldsymbol{p}_s$ is then

$$\ln(c_s) = \ln \Phi \left(\frac{\delta - \mu_1 - \mu_2}{\sqrt{\sigma_1^2 + \sigma_2^2}} \right) = \ln \Phi \left(\frac{5}{\sqrt{9}} \right) = \ln(0.95) \, . \tag{4.1}$$

An element $p_{k,j}$, for which it holds that

$$p_{k,j} = \begin{cases} 1 & \text{if job } J_j \text{ is in pattern } \boldsymbol{p}_k \\ 0 & \text{otherwise,} \end{cases} \tag{4.2}$$

then indicates whether k-th pattern schedules j-th job. We can then write

$$\max \sum_{k=1}^{|\mathcal{P}|} y_k \cdot \ln c_k \tag{4.3}$$

$$\text{subject to:} \quad \sum_{k=1}^{|\mathcal{P}|} y_k \cdot p_{k,j} \geq 1 \quad \forall j \in \{1, \ldots, N\}, \tag{4.4}$$

$$\sum_{k=1}^{|\mathcal{P}|} y_k \leq M, \tag{4.5}$$

$$\text{where:} \quad y_k \in \{0, 1\} \quad \forall k \in \{1, \ldots, |\mathcal{P}|\}. \tag{4.6}$$

The y_k is a binary decision variable that indicates if we choose the k-th pattern to be scheduled or not. The constraint (4.4) says that we cannot use more than M patterns, i.e. the number of available resources. The other set of constraints specifies that each job has to be scheduled on some machine.

The Master Problem is then a relaxation of the model (4.3) – (4.6), where we allow the y_k variables to take real positive values and the full-sized set of patterns $\mathcal{P}$ is relaxed with subset $\mathcal{P}' \subseteq \mathcal{P}$, i.e.:

$$\max \sum_{k}^{|\mathcal{P}'|} y_k \cdot \ln c_k \tag{4.7}$$

$$\text{subject to:} \quad \sum_{k=1}^{|\mathcal{P}'|} y_k \cdot p_{k,j} \geq 1 \quad \forall j \in \{1, \ldots, N\}, \tag{4.8}$$

$$\sum_{k=1}^{|\mathcal{P}'|} y_k \leq M \tag{4.9}$$

$$\text{where:} \quad y_k \in \mathbb{R}_0^+ \quad \forall k \in \{1, \ldots, |\mathcal{P}'|\} \tag{4.10}$$

4.1.1 Dual problem

Since we begin with an initial limited set of patterns, we are faced with the question which further patterns to generate. We will derive this procedure from the dual problem. The duality

principle states, that to every Linear Program (denoted as a primary problem) there exists a dual problem, such that the number of variables is the same as the number of constraints in the original one (i.e. to every constraint of original problem there is a dual variable) and the sense of the optimization is reversed (i.e. one problem is stated as maximization and the other as minimization). Furthermore, it can be shown that our problem satisfies so-called *strong duality* and both the problems are therefore equivalent.

To construct such model, we assign to each constraint (4.8) – (4.9) a dual variable:

$$\sum_{k=1}^{|\mathcal{P}'|} y_k \cdot p_{k,j} \geq 1 \quad \mapsto \quad \psi_j \leq 0, \quad \forall j \in \{1, \ldots, N\} \tag{4.11}$$

$$\sum_{k=1}^{|\mathcal{P}'|} y_k \leq M \quad \mapsto \quad \gamma \geq 0 \tag{4.12}$$

Each of these new variables will be present in the objective function, with the coefficient equal to the right side of their corresponding constraint. We invert the sense of the optimization of the primary problem and the objective is thus given as

$$\min_{\boldsymbol{\psi},\gamma} \sum_{j=1}^{N} \psi_j + M \cdot \gamma, \tag{4.13}$$

where M is the number of machines for the given instance.

The original model works with the set of variables y_k, for $k \in \{1, \ldots, |\mathcal{P}'|\}$. This implicates that our model has to have $|\mathcal{P}'|$ constraints, to which the y_k variables map. The constraints are given by:

$$\sum_{j=1}^{N} \psi_j \cdot p_{k,j} + \gamma \geq \ln c_k \quad \forall k \in \{1, \ldots, |\mathcal{P}'|\} \tag{4.14}$$

We summarize the dual problem as:

$$\min \sum_{j=1}^{N} \psi_j + M \cdot \gamma \tag{4.15}$$

$$\text{subject to:} \quad \sum_{j=1}^{N} \psi_j \cdot p_{k,j} + \gamma \geq \ln c_k \quad \forall k \in \{1, \ldots, |\mathcal{P}'|\}, \tag{4.16}$$

$$\text{where:} \quad \gamma \in \mathbb{R}_0^+, \tag{4.17}$$

$$\psi_k \in \mathbb{R}_0^- \quad \forall k \in \{1, \ldots, |\mathcal{P}'|\} \tag{4.18}$$

Note that the vector $\boldsymbol{\psi}$ is of dimension N. Therefore, each of its components corresponds to a single specific job. This vector constitutes the desired rewards for scheduling each job according to current Master Problem solution. The final question is how to obtain the initial set of patterns. This is easily solved by computing heuristic solution for the instance. From the feasible schedule it produces, we extract the patterns as scheduled on each machine. This constitutes the initial set of patterns $\mathcal{P}'$ for the Master Problem.

4.2 Pricing Problem

In this section, we derive so called Pricing Problem, that suggests which pattern to generate. We derive the objective function from the constraint given by Equation (4.16). We subtract $\ln c_k$ from both sides of the equation and get

$$\sum_{j=1}^{N} \psi_j \cdot p_{k,j} + \gamma - \ln c_k \geq 0. \tag{4.19}$$

We want to break the constraint (4.19) as much as possible. We introduce a decision variable x_j, which says if job j is to be scheduled in the newly generated pattern. Note that in this case we consider only one machine and the variables x_j represent a single vector, so the objective is to minimize the term

$$\min_{\boldsymbol{x}} -\boldsymbol{\psi}^T \boldsymbol{x} + \gamma - \ln \Phi \left(\frac{\delta - \boldsymbol{\mu}^T \boldsymbol{x}}{\sqrt{\boldsymbol{\sigma}^T \boldsymbol{x}}} \right), \tag{4.20}$$

where we substituted the definition of c_k, replaced the $p_{k,j}$ variables with the x_j assignment variables and let $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_N)$, $\boldsymbol{\sigma} = (\sigma_1^2, \sigma_2^2, \ldots, \sigma_N^2,)$, $\boldsymbol{\psi} = (-\psi_1, -\psi_2, \ldots, -\psi_N)$ and $\boldsymbol{x} = (x_1, x_2, \ldots, x_N)$. Since γ is a constant for the pricing problem, we can omit it from the objective. Further, we multiply the objective by minus one and write it as

$$\max_{\boldsymbol{x}} \ln \Phi \left(\frac{\delta - \boldsymbol{\mu}^T \boldsymbol{x}}{\sqrt{\boldsymbol{\sigma}^T \boldsymbol{x}}} \right) + \boldsymbol{\psi}^T \boldsymbol{x}, \tag{4.21}$$

which is the final form of the objective function.

The objective value of the solution of the Pricing Problem is called the *reduced cost*. With the initial set of patterns we solve the Master Problem and extract the dual solutions $\boldsymbol{\psi}$ and γ. We then pass the vector $\boldsymbol{\psi}$ to the Pricing Problem. The Pricing Problem assigns some binary value to the variables x_j. This represents the new pattern, which we add to our set of available patterns $\mathcal{P}'$. We then solve the Master Problem again and obtain the updated dual prices $\boldsymbol{\psi}$ and γ and repeat the process.

Furthermore, we make several remarks here. First, the model is non-linear and cannot be solved directly using a linear program. Moreover, the problem can be shown to be at least weakly $\mathcal{NP}$-hard by reduction from the Knapsack Problem [Štec *et al.*, to appear 2019]. All this implies that compared to the Master Problem which is linear, most of the processing time is going to be spent on the Pricing Problem. It is therefore critical not only to solve this problem, but also to give the solution as fast as possible. To this end, we provide two possible approaches to tackle this problem.

4.2.1 Mixed-Integer Linear Program for the Pricing Problem

The first approach we propose is to solve it using the already established mixed-integer linear model from the Section 3. Since the Pricing Problem considers only a single machine, we simplify the model by removing the dimension i, which identifies the different machines. Moreover, since we do not require that all the jobs are scheduled, we discard the constraint (3.41). Finally, we update the objective with the reward term from the current dual prices as

$$\max \ln \Phi(\delta \cdot t - v) + \boldsymbol{\psi}^T \boldsymbol{w}. \tag{4.22}$$

4.2.2 Variance Enumeration model

Another possible approach we propose is to fix the value $\sqrt{\boldsymbol{\sigma}^T\boldsymbol{x}} \equiv \sqrt{v}$ in the Pricing Problem objective value (4.21) and perform optimization over all the possible v values. Let $\sigma^2_{[j]}$ be an element of an ordered set $\mathcal{V}_O$ of values in $\boldsymbol{\sigma}$, such that $\forall\sigma^2_{[j]} \in \mathcal{V}_O : \sigma^2_{[j]} \geq \sigma^2_{[j+1]}$. Let $\mathcal{V} = \{\min(\boldsymbol{\sigma}), \min(\boldsymbol{\sigma}) + 1, \ldots, \sum_{j=1}^{v_{\max}} \sigma^2_{[j]}\}$. We can then transform the problem into $|\mathcal{V}|$ sub-problems as

$$\max_{v\in\mathcal{V}} \max_{\boldsymbol{x}} \log\Phi\left(\frac{\delta - \boldsymbol{\mu}^T\boldsymbol{x}}{\sqrt{v}}\right) + \boldsymbol{\psi}^T\boldsymbol{x}, \tag{4.23}$$

$$\text{subject to:} \quad v = \boldsymbol{\sigma}^T\boldsymbol{x}. \tag{4.24}$$

Since v is constant for given sub-problem, the only non-linearity is the cumulative distribution function. We can approximate that by using the *piece wise linear objective* feature provided by the Gurobi MIP solver. From all the computed solutions, we choose the one with the highest objective value. The main benefit of this approach is that each sub-problem can be solved independently of others in parallel for varying values of v.

4.3 Branching scheme

With the Master and Pricing Problem defined, we put our algorithm together. At the core it relies on an interplay of those two models. The Master Problem evaluates the jobs we have to schedule by exploiting the duality theorem, while Pricing Problem, depending on the dual costs, generates additional patterns that are inserted to the set of patterns $\mathcal{P}'$. The Master Problem is resolved again with the updated set of patterns and determines new dual costs for the jobs. Once we cannot find a pattern with a negative reduced cost, the Master Problem is solved optimally. Then, we extract the optimal schedule by looking at the variables y_k from the last solution of the Master Problem. However, we note that the Master Problem is *relaxed* and the y_k values do not necessarily have to be integers. Hence, a branching mechanism that ensures an integer solution has to be implemented.

In such cases, we add additional constraints on a pair of jobs – we create two copies of the Master Problem adding constraint that enforces that they have to be scheduled on the same machine and the opposite constraint stating that they must be scheduled separately to the other copy. We then run the whole algorithm again, with the addition of these constraints to the Pricing Problem solver, that must respect them in the new pattern generation. If the next iteration of the solver again stops with an non-integer solution, we need to apply the constraints to some other two jobs again.

This procedure creates a branching tree, where each node represents a single Master Problem with constraints given by the current and parent nodes up to the root. If in some node we find an integer solution, we stop the execution and close the node. Otherwise, we need to branch and continue until an integer solution is found. Once all the leaves are closed, we take the best solution we could find as it represents the optimal schedule. We denote a single such branch as a *branching decision*.

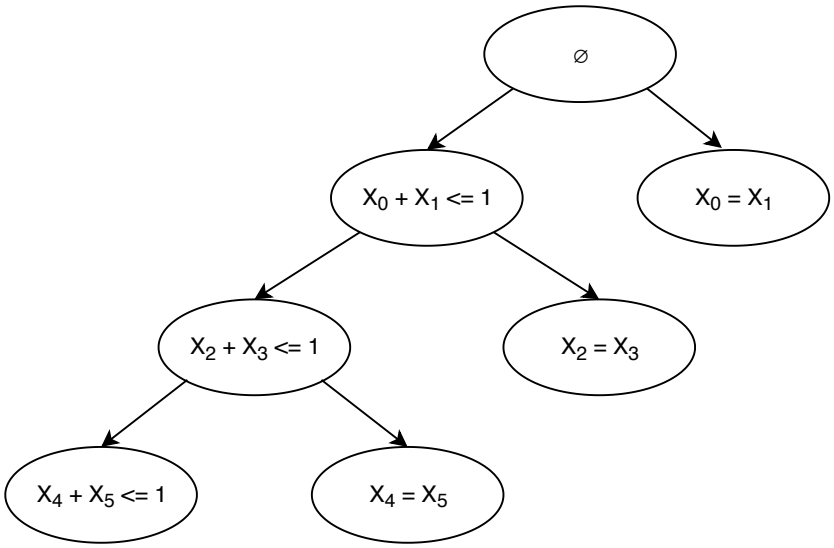


Figure 3: Example branching tree.

Figure 3 depicts an example branching tree, where each node represents a single Master Problem. Constraints of type $X_i = X_j$ indicates that the jobs J_i and J_j have to be scheduled together on some machine. Constraints written as $X_i + X_j \leq 1$ mean that they cannot be assigned to the same machine. The node labeled as $\emptyset$ indicates the initial node where no branching constraints are enforced.

To make the branching tree as balanced as possible, to each job J_j we assign a variable D_j, called a constraint degree of the job. Its value corresponds to the number of constraints generated by this branching scheme, where the given job plays a role. Then we create an ordered set of all the jobs, denoted as $\mathcal{J}_D$, where we sort the jobs with respect to their constraint degree in ascending order. In this way, the jobs on which we branch the least will always be at the front.

To add new pair of constraints we simply take the first two jobs in this ordered set $\mathcal{J}_D$. Here we can run into two possible issues. The first is that we can try to create a constraint that already exists. This can be simply checked and if we were about to do it, we simply replace the second job with the one next in order. If the situation repeats, we again take the next following job.

The second issue are the conflicts in the generated constraints. As an example, assume we have 2 constraints at the current Master Problem - that job J_1 and job J_2 have to be scheduled on the same machine and that job J_2 and J_3 have to be scheduled on the same machine. Obviously, if we were to add the constraint saying that job J_1 and job J_3 cannot be scheduled on the same machine, we would have a contradiction in our hands.

To prevent this, we introduce two safeguards. The first one is a conflict avoidance mechanism. Each job is represented as a single node of a graph. Each constraint on their pairing is then represented as an edge. If it corresponds to enforced pairing (the jobs must be scheduled on the same machine), the two nodes may be seen as a single one (i.e. the jobs are merged into one). The other constraint (must not be together) is just a simple edge. When we are about to add a constraint, we first perform a check whether any conflict would be formed. This can be seen as a variation of the *union find problem*. If it would, we take look at the

next variable in the ordered set J_D and try to apply constraint there. The whole procedure is shown in Algorithm 2, where $\mathcal{B}$ is a set of elements defined as tuple (a, b, e), where e is $true$ if jobs a and b must be scheduled on the same machine and $false$ otherwise. $\mathcal{B}$ therefore represents the set of non-conflicting branching decisions.

Algorithm 2 Conflict avoidance mechanism

```
d ← {0}^N
B ← ∅
while some branching node still open do
    run Branch-and-Price
    if integer solution found then
        close node
    else
        J_D ← sort J with respect to d
        i_1 ← 0
        i_2 ← 1
        while (J_D[i_1], J_D[i_2]) ∈ B ∨ unionFind(J_D[i_1], J_D[i_2]) do
            i_2 ← i_2 + 1
            if i_2 ≥ N then
                i_1 ← i_1 + 1
                i_2 ← i_1 + 1
            end if
        end while
        d[i_1] ← d[i_1] + 1
        d[i_2] ← d[i_2] + 1
        branch on J_D[i_1], J_D[i_2]
    end if
end while
```

If we come to the point where we need to introduce a constraint on the pairing of jobs, we need to remove the already existing patterns that violate it. This in turn can cause an infeasibility of the Master Problem model, since we can end up with insufficient number of patterns (i.e. less than the number of machines) or none of the remaining patterns would schedule some subset of jobs. So after we prune the existing patterns, we need to check for the feasibility of the model and if we cannot proceed, with need to regenerate additional patterns with respect to the given constraints. A check for the feasibility of the Master problem and additional pattern generation is done by the second safeguard, which is called *the recovery model* and is described in the following section.

4.3.1 Recovery model

The purpose of the recovery model is to test the feasibility of the model under given branching decisions. Additionally, the recovery model generates additional patterns for the Master Problem, if the current set proves to be insufficient. Let $x_{i,j}$ be a binary assignment

variable for $i \in \{1, \ldots, M\}, j \in \{1, \ldots, N\}$. Then we formulate the objective function as a minimization of the sum across all these variables:

$$\min \sum_{i=1}^{M} \sum_{j=1}^{N} x_{i,j} \tag{4.25}$$

The model has to schedule all the jobs, the same as the original model, so we introduce the constraint:

$$\sum_{i=1}^{M} x_{i,j} = 1. \tag{4.26}$$

Furthermore, we need to ensure that the model takes into the account all the pairing constraints introduced by the branching scheme.

Now consider a general case where we have M machines and N jobs. Furthermore assume a single pairing constraint on jobs J_1 and J_2 saying that they must be scheduled together. Therefore, nothing prevents the model to create a solution where a first pattern is a unit vector $\mathbf{1}$ and all other $M-1$ patterns are zero vector $\mathbf{0}$. Similarly, for a case where we would prohibit the paired scheduling of those two jobs, we might end up with a pattern that does not schedule only this single job and then a pattern that only schedules this job, with the rest of the patterns again being zero vectors.

To prevent such disproportionate distribution of jobs in patterns, we introduce the constraint stating that no pattern can schedule more than v_{max} (introduced in Section 2) jobs:

$$\sum_{j=1}^{N} x_{i,j} \leq v_{max} \quad \forall i \in \{1, \ldots, M\} \tag{4.27}$$

This will ensure that the generated patterns are more balanced. To summarize, the complete model is given by the following integer linear program:

$$\min \sum_{i=1}^{M} \sum_{j=1}^{N} x_{i,j} \tag{4.28}$$

$$\text{subject to:} \quad \sum_{i=1}^{M} x_{i,j} = 1 \quad \forall j \in \{1, \ldots, N\}, \tag{4.29}$$

$$\sum_{j=1}^{N} x_{i,j} \leq v_{max} \quad \forall i \in \{1, \ldots, M\}, \tag{4.30}$$

$$x_{a,i} = x_{b,i} \quad \forall (a, b, true) \in \mathcal{B}, \forall i \in \{1, \ldots, M\}, \tag{4.31}$$

$$x_{a,i} + x_{b,i} \leq 1 \quad \forall (a, b, false) \in \mathcal{B}, \forall i \in \{1, \ldots, M\} \tag{4.32}$$

$$\text{where:} \quad x_{i,j} \in \{0, 1\} \quad \forall i \in \{1, \ldots, M\}, \forall j \in \{1, \ldots, N\}. \tag{4.33}$$

The recovery model performs a test run of the problem with current branching decisions and tests its feasibility. If some conflict in the introduced branching decisions is present, the model fails to produce a result and we cut the node off of the branching tree.

5 Two-machines problem

In this section, we introduce an algorithm for a special case of the problem with exactly two parallel machines. We first reformulate the problem statement for this special case. Then we show that the *relaxation* of this problem leads to maximization of a concave function of a single variable. Finally, we show that from the solution of the relaxed problem we obtain the optimal schedule by solving two sub-problems.

5.1 Reformulation

Let $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_N)$, $\boldsymbol{\sigma} = (\sigma_1^2, \sigma_2^2, \ldots, \sigma_N^2)$ and $\boldsymbol{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,N})$ be vectors of means, variances and job-machine assignment variables $x_{i,j}$. We begin by writing the objective function for the case of two machines. We get:

$$\max_{\boldsymbol{x}_1, \boldsymbol{x}_2} \prod_{i=1}^{2} \Phi\left(\frac{\delta - \boldsymbol{\mu}^T \boldsymbol{x}_i}{\sqrt{\boldsymbol{\sigma}^T \boldsymbol{x}_i}}\right) \tag{5.1}$$

By applying the logarithm and expanding the sum we write that

$$\max_{\boldsymbol{x}_1, \boldsymbol{x}_2} \ln \Phi\left(\frac{\delta - \boldsymbol{\mu}^T \boldsymbol{x}_1}{\sqrt{\boldsymbol{\sigma}^T \boldsymbol{x}_1}}\right) + \ln \Phi\left(\frac{\delta - \boldsymbol{\mu}^T \boldsymbol{x}_2}{\sqrt{\boldsymbol{\sigma}^T \boldsymbol{x}_2}}\right). \tag{5.2}$$

Since we schedule all the given tasks, the vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ have to be mutually exclusive on a corresponding components. We can therefore write that

$$\boldsymbol{\mu}^T \boldsymbol{x}_2 = \boldsymbol{\mu}^T \mathbf{1} - \boldsymbol{\mu}^T \boldsymbol{x}_1, \tag{5.3}$$

$$\boldsymbol{\sigma}^T \boldsymbol{x}_2 = \boldsymbol{\sigma}^T \mathbf{1} - \boldsymbol{\sigma}^T \boldsymbol{x}_1, \tag{5.4}$$

where $\mathbf{1}$ is the unit vector. In other words, the sum of means on one machine is equal to the total sum of all available means minus the sum of the means on the other machine. Let $\bar{v} = \boldsymbol{\sigma}^T \mathbf{1}$ and $\mathcal{V} = \{\min(\boldsymbol{\sigma}), \min(\boldsymbol{\sigma}) + 1, \ldots, \lceil \frac{\bar{v}}{2} \rceil\}$. We then reformulate the problem as

$$\max_{v \in \mathcal{V}} \max_{\boldsymbol{x}} \ln \Phi\left(\frac{\delta - \boldsymbol{\mu}^T \boldsymbol{x}}{\sqrt{v}}\right) + \ln \Phi\left(\frac{\delta - \boldsymbol{\mu}^T \mathbf{1} + \boldsymbol{\mu}^T \boldsymbol{x}}{\sqrt{\bar{v} - v}}\right) \tag{5.5}$$

$$\text{subject to:} \quad \boldsymbol{\sigma}^T \boldsymbol{x} = v \tag{5.6}$$

$$\text{where:} \quad \boldsymbol{x} \in \{0, 1\}^N, \tag{5.7}$$

where N is the number of jobs to schedule. We will solve the problem similarly as in Section 4.2.2. It is sufficient to test the values up to $\lceil \frac{\bar{v}}{2} \rceil$ due to the symmetry of the two machines. For each fixed $v \in \mathcal{V}$, we solve the problem only in terms of $\boldsymbol{x}$ variables. We show how in the following section.

5.2 Relaxation

We fix the value v and get a sub-problem which we solve from its relaxation. We will use the substitutions

$$a \equiv \delta - \boldsymbol{\mu}^T \mathbf{x}, \tag{5.8}$$

$$c \equiv -2 \cdot \delta + \boldsymbol{\mu}^T \mathbf{1} \tag{5.9}$$

and the following identity:

$$\Phi(z) = 1 - \Phi(-z), \quad z \in \mathbb{R} \tag{5.10}$$

We then write the relaxed problem as

$$\max_a \ln \Phi\left(\frac{a}{\sqrt{v}}\right) + \ln\left(1 - \Phi\left(\frac{c+a}{\sqrt{\bar{v}-v}}\right)\right) \tag{5.11}$$

$$\text{where:} \quad a \in \mathbb{R}. \tag{5.12}$$

Further, using the definition of the normal cumulative distribution function, we get that

$$\Phi(x) = \int_{-\infty}^{x} \phi(t)dt = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt, \tag{5.13}$$

where $\phi(x)$ is the normal probability density function. We then write the objective as

$$\ln\left(\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{a}{\sqrt{v}}} e^{-\frac{t^2}{2}} dt\right) + \ln\left(1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{c+a}{\sqrt{\bar{v}-v}}} e^{-\frac{t^2}{2}} dt\right) = g(a). \tag{5.14}$$

Note that $g(a)$ is a concave function.

We now wish to find a maximum of this function, which is the problem of finding an extrema of a function:

$$\frac{dg}{da} = 0 \tag{5.15}$$

Note the function is only a function of a variable a, since we hold the v variable fixed. Moreover, the a variable is only present in the bound of the integral. To find a derivative of such function with respect to a, we use the *Leibniz integral rule*, that states the following:

$$\frac{d}{dx}\left(\int_{a(x)}^{b(x)} f(x,t)dt\right) = f(x,b(x)) \cdot \frac{d}{dx}b(x) - f(x,a(x)) \cdot \frac{d}{dx}a(x) + \int_{a(x)}^{b(x)} \frac{\partial}{\partial x} f(x,t)dt \tag{5.16}$$

Since our function f inside the integral is the probability density function $\phi(t)$ and the lower bound of the integral is a constant, the second and third term evaluate to zero:

$$\int_{-\infty}^{a} \frac{d}{da}\phi(t)dt = 0 \tag{5.17}$$

$$\phi(t) \cdot \frac{d}{dx}(-\infty) = 0. \tag{5.18}$$

Now to find the derivative, we get

$$\frac{d}{da}g(a) = \frac{1}{\Phi\left(\frac{a}{\sqrt{v}}\right)} \frac{d}{da}\Phi\left(\frac{a}{\sqrt{v}}\right) - \frac{1}{1 - \Phi\left(\frac{c+a}{\sqrt{\bar{v}-v}}\right)} \frac{d}{da}\Phi\left(\frac{c+a}{\sqrt{\bar{v}-v}}\right). \tag{5.19}$$

We then find the corresponding derivatives using the Equation (5.16):

$$\frac{d}{da}\Phi\left(\frac{a}{\sqrt{v}}\right) = \phi\left(\frac{a}{\sqrt{v}}\right)\cdot\frac{1}{\sqrt{v}} \tag{5.20}$$

$$\frac{d}{da}\Phi\left(\frac{c+a}{\sqrt{\bar{v}-v}}\right) = \phi\left(\frac{c+a}{\sqrt{\bar{v}-v}}\right)\cdot\frac{1}{\sqrt{\bar{v}-v}} \tag{5.21}$$

So we rewrite the derivative as

$$\frac{d}{da}g(a) = \frac{\phi\left(\frac{a}{\sqrt{v}}\right)}{\Phi\left(\frac{a}{\sqrt{v}}\right)}\cdot\frac{1}{\sqrt{v}} - \frac{\phi\left(\frac{c+a}{\sqrt{\bar{v}-v}}\right)}{1-\Phi\left(\frac{c+a}{\sqrt{\bar{v}-v}}\right)}\cdot\frac{1}{\sqrt{\bar{v}-v}} = 0. \tag{5.22}$$

We then multiply both sides with the term $1-\Phi\left(\frac{c+a}{\sqrt{\bar{v}-v}}\right) > 0$,

$$\xi\cdot\frac{\phi\left(\frac{a}{\sqrt{v}}\right)}{\sqrt{v}} - \phi\left(\frac{c+a}{\sqrt{\bar{v}-v}}\right)\cdot\frac{1}{\sqrt{\bar{v}-v}} = 0, \tag{5.23}$$

where

$$\xi \equiv \frac{1-\Phi\left(\frac{c+a}{\sqrt{\bar{v}-v}}\right)}{\Phi\left(\frac{a}{\sqrt{v}}\right)}. \tag{5.24}$$

We conjecture, that Equation (5.22) is analytically unsolvable due to presence of a variable a both in the cumulative distribution function Φ and the probability distribution function ϕ. However, we show that for our test cases, it holds that $\xi \approx 1$. Figure 4 shows a plot of the function $\xi(a)$ for fixed value of c and varying v parameter. As it can be seen, the ξ value is close to 1, with the exception of the borders of the interval. By assuming $\xi = 1$ we obtain a

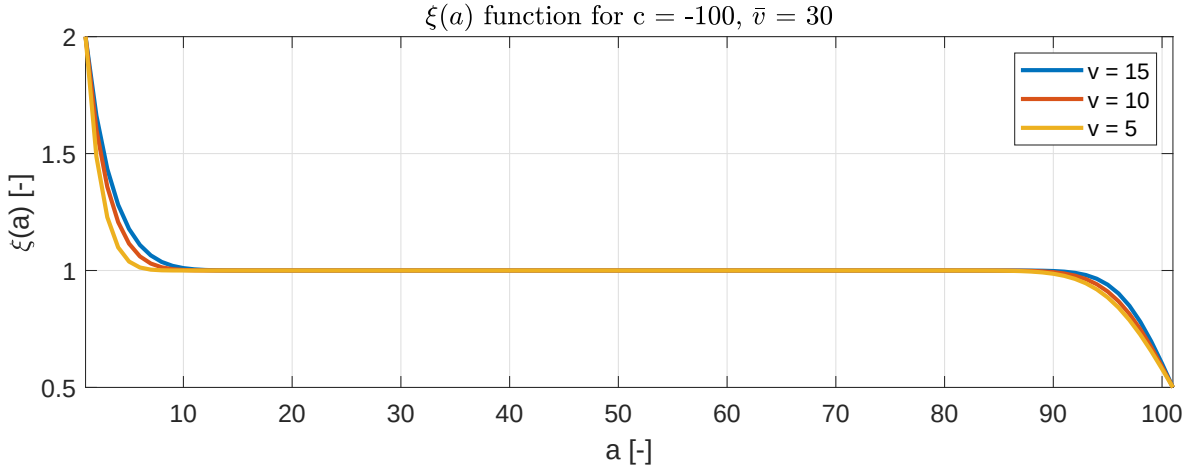


Figure 4: Example plot of $\xi(a)$ function.

surrogate equation:

$$\frac{1}{\sqrt{v}}\cdot\phi\left(\frac{a}{\sqrt{v}}\right) - \frac{1}{\sqrt{\bar{v}-v}}\cdot\phi\left(\frac{c+a}{\sqrt{\bar{v}-v}}\right) = 0 \tag{5.25}$$

We then employ the definition of the normal probability density function $\phi(t)$:

$$\frac{1}{\sqrt{2\pi}}e^{-\frac{a^2}{2v}} = \frac{1}{\sqrt{2\pi}}e^{-\frac{(c+a)^2}{2(\bar{v}-v)}} \cdot \frac{\sqrt{v}}{\sqrt{\bar{v}-v}} \tag{5.26}$$

By multiplying by the constant $\sqrt{2\pi}$ and applying the natural logarithm to both sides we get that

$$-\frac{a^2}{2v} = -\frac{(c+a)^2}{2(\bar{v}-v)} + \frac{1}{2}\ln\left(\frac{v}{\bar{v}-v}\right). \tag{5.27}$$

We continue expanding the term given by the Equation (5.27). Multiplying the equation by -2 we get that

$$\frac{a^2}{v} = \frac{(c+a)^2}{(\bar{v}-v)} - \ln\left(\frac{v}{\bar{v}-v}\right). \tag{5.28}$$

We expand the term $(c+a)^2$ and get the following:

$$\frac{a^2+2ac+c^2}{(\bar{v}-v)} - \ln\left(\frac{v}{\bar{v}-v}\right) = \frac{a^2}{v} \tag{5.29}$$

$$\frac{a^2}{v} - \frac{a^2}{(\bar{v}-v)} - \frac{2ac}{(\bar{v}-v)} - \frac{c^2}{(\bar{v}-v)} + \ln\left(\frac{v}{\bar{v}-v}\right) = 0 \tag{5.30}$$

$$a^2 \cdot \left(\frac{1}{v} - \frac{1}{\bar{v}-v}\right) - a \cdot \frac{2c}{\bar{v}-v} - \frac{c^2}{(\bar{v}-v)} + \ln\left(\frac{v}{\bar{v}-v}\right) = 0 \tag{5.31}$$

This is a quadratic equation with variable a and we solve it using the standard formula. First, we evaluate the determinant D:

$$D = \frac{4c^2}{(\bar{v}-v)^2} - 4 \cdot \left(\frac{1}{v} - \frac{1}{\bar{v}-v}\right) \cdot \left(-\frac{c^2}{\bar{v}-v}\right) - 4 \cdot \left(\frac{1}{v} - \frac{1}{\bar{v}-v}\right) \cdot \ln\left(\frac{v}{\bar{v}-v}\right) \tag{5.32}$$

To proceed further, we calculate that

$$\frac{1}{v} - \frac{1}{\bar{v}-v} = \frac{\bar{v}-2v}{v \cdot (\bar{v}-v)}, \tag{5.33}$$

so the second term of the determinant evaluates to

$$\left(\frac{1}{v} - \frac{1}{\bar{v}-v}\right) \cdot \left(-\frac{c^2}{\bar{v}-v}\right) = \frac{-c^2 \cdot (\bar{v}-2v)}{v \cdot (\bar{v}-v)^2}. \tag{5.34}$$

We substitute this into the determinant and do additional changes:

$$D = 4 \cdot \left(\frac{c^2}{(\bar{v}-v)^2} \cdot \left(1 + \frac{\bar{v}-2v}{v}\right) - \left(\frac{1}{v} - \frac{1}{\bar{v}-v}\right) \cdot \ln\left(\frac{v}{\bar{v}-v}\right)\right) \tag{5.35}$$

Finally, since we can write

$$1 + \frac{\bar{v}-2v}{v} = 1 + \frac{\bar{v}-v}{v} - \frac{v}{v} = \frac{\bar{v}-v}{v}, \tag{5.36}$$

we get the final formula for the determinant:

$$D = 4 \cdot \left(\frac{c^2}{(\bar{v} - v) \cdot v} - \left(\frac{1}{v} - \frac{1}{\bar{v} - v} \right) \cdot \ln \left(\frac{v}{\bar{v} - v} \right) \right) \tag{5.37}$$

After the substitution into the quadratic root formula, we get that

$$a^*_{1,2} = \frac{\frac{c}{\bar{v}-v} \pm \sqrt{\frac{c^2}{(\bar{v}-v)\cdot v} - \left(\frac{1}{v} - \frac{1}{\bar{v}-v} \right) \cdot \ln \left(\frac{v}{\bar{v}-v} \right)}}{\frac{1}{v} - \frac{1}{\bar{v}-v}}. \tag{5.38}$$

$a^*_{1,2}$ are solutions for the surrogate Equation (5.25). However, since ξ is not exactly 1, we need to find the true global optimum of (5.11) by a gradient descent procedure.

5.3 Complete algorithm

The Equation (5.38) gives us two real solutions to the surrogate problem, but only one solution is valid, because the second solution was introduced due to the approximation of the ξ value. Figure 5 shows the plot of the original function (5.22) and the surrogate function (5.25). Note, that the surrogate function (5.25) is a subtraction of two normal distributions, divided by different coefficients.

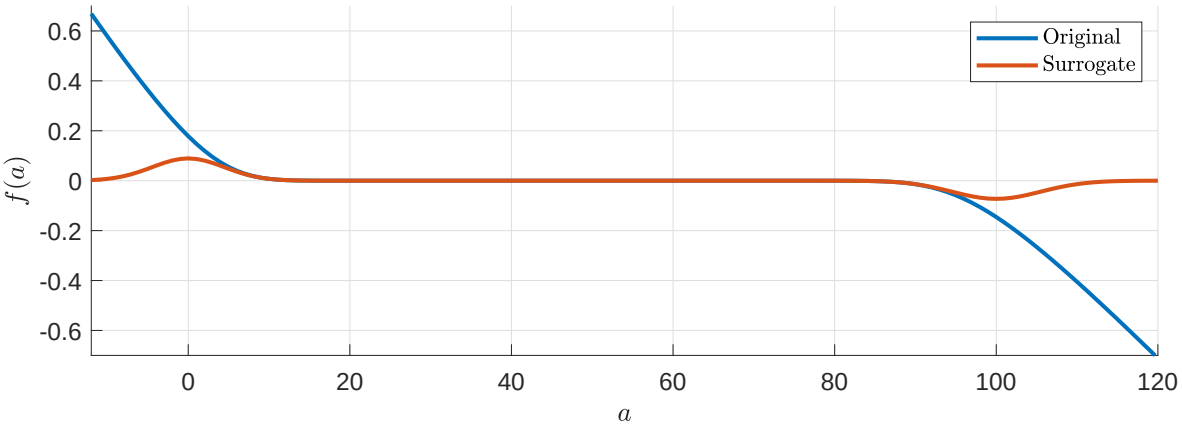


Figure 5: Plot of Equation (5.22) and Equation (5.25).

The first root of the quadratic equation is located between the two normal distributions as seen in Figure 5. The second root is a result of the two distributions converging to zero at different ratios from different sides of the x-axis. Since we set $1 \leq v \leq \lceil \frac{\bar{v}}{2} \rceil$, it holds that

$$\frac{1}{\sqrt{v}} \geq \frac{1}{\sqrt{\bar{v} - v}}. \tag{5.39}$$

This implies that the first distribution decreases at higher rate than the other one. Therefore at some point, where $a \leq 0$, the slower decrease of the second distribution overtakes the faster decrease of the first one and drags the whole functional value below zero, resulting in second root. This is contrary to the original equation, where the ξ term drives the whole function to

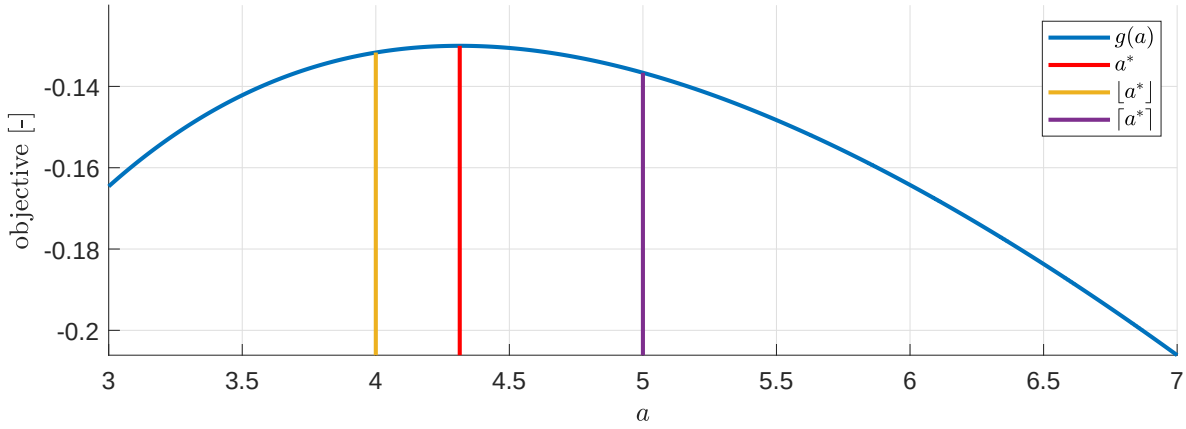


Figure 6: Plot of the objective function with highlighted real solution for the maximization problem ($c = -13, \bar{v} = 50, v = 5$).

infinity. Only the root a_2^* is valid and we set it as the starting point of the gradient descent procedure, since a_2^* is very close to the true global optimum of (5.11).

To solve the original problem, we must find such integer value of a that maximizes the objective function. First we improve the solution by performing a gradient descent to maximize the relaxed objective and get maximal value a^*. We need to do this step, since we performed estimation of the ξ value. Then, since the $g(a)$ is a concave function, we know that optimal integer solution is going to be located on the interval $[\lfloor a^* \rfloor, \lceil a^* \rceil]$. Figure 6 depicts this situation. We then find the corresponding pattern by solving two mixed-integer linear programs given as

$$
\begin{aligned}
&\text{MILP}^+(a^*, v): \\
&\qquad\qquad \min a \\
&\text{subject to:} \quad a \geq \lceil a^* \rceil, \\
&\qquad\qquad a = \delta - \boldsymbol{\mu}^T \boldsymbol{x}, \\
&\qquad\qquad v = \boldsymbol{\sigma}^T \boldsymbol{x} \\
&\text{where:} \quad a \in \mathbb{Z}, \\
&\qquad\qquad \boldsymbol{x} \in \{0,1\}^N
\end{aligned}
\qquad\qquad
\begin{aligned}
&\text{MILP}^-(a^*, v): \\
&\qquad\qquad \max a \\
&\text{subject to:} \quad a \leq \lfloor a^* \rfloor \\
&\qquad\qquad a = \delta - \boldsymbol{\mu}^T \boldsymbol{x} \\
&\qquad\qquad v = \boldsymbol{\sigma}^T \boldsymbol{x} \\
&\text{where:} \quad a \in \mathbb{Z}, \\
&\qquad\qquad \boldsymbol{x} \in \{0,1\}^N
\end{aligned}
$$

The outputs of the linear programs are two patterns $\boldsymbol{x}_+^*, \boldsymbol{x}_-^*$. We consider only a single pattern that generates schedule with the better service level. The whole algorithm is shown in Algorithm 3.

Algorithm 3 Two-machines model

$h_{\text{best}} \leftarrow 0, \boldsymbol{x}_{\text{best}} \leftarrow \emptyset$
for v **in** $\mathcal{V}$ **do**
  $a^* \leftarrow \frac{\frac{c}{\bar{v}-v} - \sqrt{\frac{c^2}{(\bar{v}-v)\cdot v} - \left(\frac{1}{v} - \frac{1}{\bar{v}-v}\right)\cdot \ln\left(\frac{v}{\bar{v}-v}\right)}}{\frac{1}{v} - \frac{1}{\bar{v}-v}}$
  while $g(a^*) < \max(g(a^*-1), g(a^*+1))$ **do**
    if $g(a^*) < g(a^*-1)$ **then**
      $a^* \leftarrow a^* - 1$
    else
      $a^* \leftarrow a^* + 1$
    end if
  end while
  $\boldsymbol{x}^*_+ \leftarrow \text{MILP}^+(a^*, v)$
  $\boldsymbol{x}^*_- \leftarrow \text{MILP}^-(a^*, v)$
  $\boldsymbol{x}^* \leftarrow \arg\max(h(\boldsymbol{x}^*_+), h(\boldsymbol{x}^*_-))$
  if $h(\boldsymbol{x}^*) > h_{\text{best}}$ **then**
    $h_{\text{best}} \leftarrow h(\boldsymbol{x}^*)$
    $\boldsymbol{x}_{\text{best}} \leftarrow \boldsymbol{x}^*$
  end if
end for
return $h_{\text{best}}, \boldsymbol{x}_{\text{best}}$

The $h(\boldsymbol{x})$ is a function that calculates the service level from schedule generated by pattern $\boldsymbol{x}$.

6 Experimental results

In this section, we provide computational experiments for the methods proposed in the thesis.

6.1 Experimental setup

To compare our proposed algorithms, we implemented the better one of two exact Branch-and-Bound algorithms proposed by [Ranjbar *et al.*, 2012] (named B&B1) as a reference. All the methods (with the exception of the Two-machines model and the Non-linear model) were implemented in C++. The Two-machines and Non-linear model were implemented in Python. We used the Gurobi 8.0 solver to solve the mixed-integer linear program models. The Non-linear model was solved using the SCIP 6.0.0 solver.

All experiments were run on a server with two Intel Xeon E5-2620 v4 processors, each having 28 cores, 252 GB RAM memory and a 64-bit operating system. For the Mixed-Integer Linear Program and Branch-and-Price experiments, 16 cores of one CPU were allocated. Since the reference Branch-and-Bound implementation does not perform any computations in parallel, only a single core was allocated. A single core was allocated for the Non-linear model as well. The Two-machines model was separately tested using 1, 4, 8 and 16 cores to observe the speed-up effect of the increased number of CPUs available.

All instances were generated using the method described by [Ranjbar *et al.*, 2012]. The exact value of the parameters used for the instance generation varied depending on the tested model.

The non-linear model was tested using set of instances with 2, 3 and 4 machines and $N \in \{10, 11, 12, 13, 14, 15\}$ jobs. The c parameter (as defined by [Ranjbar *et al.*, 2012], influences the variance of the distribution which generates the job variances) was set to either 0.25 or 0.75. For each such combination (M, N, c) 10 instances were generated, with total of 240 instances in the whole test set.

The Mixed-Integer Linear Program instance set was generated using fixed values of the number of machines and the number of jobs, namely $M = 3, N = 12$. The only variable was the c parameter, which took the values of $0.25, 0.50, 0.75$ and 1.00. For each such triplet $(3, 12, c)$, 10 instances were generated, with 40 instances in the whole set in total.

For the Branch-and-Price, we created instances with 2, 3, 4, 5 and 6 machines and 14, 16, 18 and 20 jobs. The c parameter was set either to 0.25 or 0.75 . For each such triplet (M, N, c), 10 instances were generated, giving a total of 400 instances. Heuristic methods were tested on the same test suite.

The subset of the Branch-and-Price instances where $M = 2$ was used for the comparison of the reference Branch-and-Bound, proposed Branch-and-Price and the Two-machines model. This set was then extended with instances having 40, 80, 200 and 500 jobs, which were also solved by the Two-machines model, Branch-and-Price and Branch-and-Bound. This was done to showcase the scaling properties of the model. We also calculate the corresponding ξ value for each optimal solution to instances from this set, to see if our approximation of Equation (5.22) is justified.

For each tested algorithm we set the timeout of an one hour, after which if the program did not finish its computation, it was stopped. Such instances, where we were not able to

find or prove an optimal solution in the given time, were not included in the mean runtime and expanded nodes calculation. We however provide percentage of such instances with each method.

6.2 Heuristic performance

Figure 7 shows a histogram of objective values calculated using the initial heuristic proposed by [Ranjbar *et al.*, 2012] and of values obtained using our LJAF heuristic.

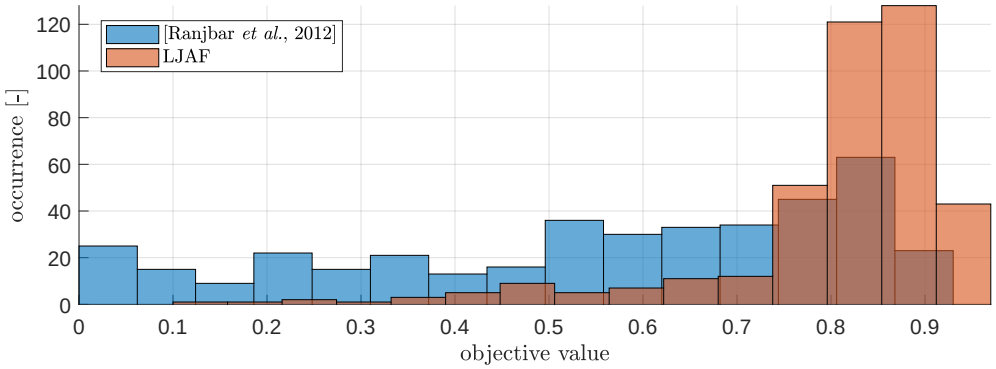


Figure 7: Histogram showing comparison of initial heuristics objective values.

Since both approaches give a feasible schedule, their objective values serve as a lower bound on the objective. It is clear that our approach is more likely to produce a tighter lower bound and is therefore better suited for the problem.

6.3 Non-linear solver performance

Table 2 shows a comparison of the Non-linear model run against the reference Branch-and-Bound. The non-linear model suffers from the same problem of finding symmetrical solutions as the reference algorithm. To combat this issue we introduced a symmetry-breaking constraint stating that the sum of mean values on the i-th machine has to be greater or equal than the sum of means on the $(i+1)$-th machine, i.e. $\mu_{M_i} \geq \mu_{M_{i+1}}, \forall i \in \{1, \ldots, M-1\}$.

The columns 3 and 4 show the performance of the Non-linear model with the symmetry-breaking constraint, the next two columns the same model but without the symmetry-breaking constraint, the last columns show results for the reference Branch-and-Bound algorithm. For each approach we give the mean runtime and the mean number of expanded nodes together with their standard deviation.

We note that the symmetry-breaking constraint had significant effect on the performance of the non-linear model. On the largest tested instances the speed-up was approximately seven-fold. This is also reflected in a lower number of expanded nodes for each instance. Compared to the Branch-and-Bound algorithm the non-linear model does not perform well. On the smallest instances ($M = 2, N \in \{10, 11, 12\}$) the reference algorithm was able to solve them in

less than 100 milliseconds. On the largest instances ($M = 3, N = 15$ and $M = 4, N = 15$) the reference algorithm still outperformed the non-linear solver, although the number of expanded nodes is significantly higher in every case.

Table 2: Comparison of Non-linear solver and the reference Branch-and-Bound.

		Non-linear model		Non-linear model (No symmetry break)		Branch-and-Bound [Ranjbar *et al.*, 2012]	
machines	jobs	runtime [s]	nodes [–]	runtime [s]	nodes [–]	runtime [s]	nodes [–]
$M = 2$	$N = 10$	1.1 (±0.4)	76.0 (±44.1)	1.4 (±0.4)	232.8 (±191.6)	0.0 (±0.0)	503.8 (±64.2)
	$N = 11$	1.0 (±0.4)	108.3 (±53.5)	1.8 (±0.6)	353.1 (±242.1)	0.0 (±0.0)	1.0K (±126.6)
	$N = 12$	1.3 (±0.5)	160.2 (±97.1)	1.9 (±0.4)	393.6 (±199.0)	0.0 (±0.0)	1.9K (±246.9)
	$N = 13$	1.3 (±0.7)	255.8 (±200.8)	2.9 (±1.0)	1.0K (±609.3)	0.1 (±0.0)	4.0K (±495.4)
	$N = 14$	1.5 (±0.8)	294.6 (±268.1)	2.8 (±1.1)	1.1K (±1.2K)	0.2 (±0.0)	8.3K (±803.5)
	$N = 15$	2.1 (±1.1)	746.4 (±735.3)	3.7 (±1.3)	1.3K (±610.8)	0.4 (±0.0)	16.4K (±1.2K)
$M = 3$	$N = 10$	3.8 (±1.4)	503.8 (±261.0)	6.6 (±2.2)	2.3K (±1.3K)	0.1 (±0.0)	2.5K (±613.5)
	$N = 11$	5.1 (±1.4)	977.9 (±509.1)	13.0 (±8.6)	4.7K (±4.0K)	0.2 (±0.0)	8.5K (±1.6K)
	$N = 12$	8.5 (±2.9)	2.1K (±863.2)	25.4 (±11.9)	9.5K (±4.9K)	0.6 (±0.1)	24.1K (±4.7K)
	$N = 13$	14.9 (±5.3)	4.5K (±2.4K)	55.7 (±28.2)	21.9K (±12.8K)	1.8 (±0.4)	71.2K (±14.2K)
	$N = 14$	35.5 (±28.9)	14.8K (±13.0K)	207.3 (±188.0)	66.7K (±50.9K)	6.0 (±1.3)	236.5K (±53.1K)
	$N = 15$	70.4 (±41.3)	29.0K (±18.5K)	586.5 (±534.3)	139.5K (±101.5K)	18.5 (±3.8)	729.9K (±157.1K)
$M = 4$	$N = 10$	5.9 (±2.5)	1.1K (±649.0)	19.8 (±9.3)	6.5K (±3.2K)	0.1 (±0.0)	2.7K (±971.0)
	$N = 11$	9.7 (±2.2)	2.6K (±1.1K)	58.9 (±36.8)	19.1K (±12.6K)	0.3 (±0.1)	9.4K (±2.5K)
	$N = 12$	20.3 (±11.0)	6.9K (±5.0K)	129.6 (±107.9)	40.1K (±34.2K)	1.3 (±0.4)	42.1K (±13.1K)
	$N = 13$	50.0 (±38.4)	16.9K (±12.5K)	390.8 (±319.6)	123.5K (±98.0K)	5.3 (±1.6)	172.1K (±56.0K)
	$N = 14$	112.2 (±75.5)	37.7K (±24.0K)	750.9 (±552.2)	204.1K (±164.7K)	23.3 (±7.2)	766.5K (±237.8K)
	$N = 15$	267.0 (±131.8)	79.0K (±40.2K)	1.3K (±979.3)	304.4K (±149.6K)	91.9 (±18.0)	3.0M (±589.1K)

6.4 Mixed-Integer Linear Program performance

Mixed-Integer Linear Program (MILP) was tested on 40 instances with fixed parameters $M = 3$ and $N = 12$. Table 3 shows the results of the test run of the MILP approach. The first column represents the resolution of the approximation interval for the division approximation (denoted as ΔH, see Section 3.4.1), the second column the resolution of the approximation interval for the objective function (denoted as ΔF, see Section 3.4.2). The next three columns show the values calculated from the test run of those 40 instances for the given pair of precision parameters. The first one shows the mean runtime in seconds with corresponding standard deviation. The second column shows the percentage (out of 40) of instances where the actual optimal solution was found and the last one informs about the mean percentual gap of the MILP objective function from the actual optimum for the non-optimally solved instances.

The results of the MILP were compared to the results of the reference Branch-and-Bound. However, since the instances were rather homogeneous (fixed M, N) we use a single reference runtime given as $t_{BB} = 0.02$ seconds, which expresses the mean runtime needed by the reference algorithm to solve the instances to optimum. There are several remarks to be made here. First, it is clear that even on the lowest approximation resolution the model lacks in the runtime performance compared to the reference algorithm. This was expected and is the reason why we did not expand the test set further. The second thing to note is that even with the highest degree of approximation, the linear approximation did not find the optimal schedule and objective value in all the cases.

Further, it appears that the precision of the division approximation does not influence the resulting objective value significantly, but influences the total runtime greatly. This is in contrast to the objective function approximation resolution, which with smoother approxima-

Table 3: Performance of the Mixed-Integer Linear Program model.

ΔH	ΔF	Runtime [s]	Optimally solved [%]	Objective gap [%]
2	1	6.31 (±2.7)	7.5	4.1704
	0.5	6.79 (±3.3)	35	7.7836
	0.2	8.41 (±4.1)	67.5	0.9431
	0.1	7.78 (±3.5)	72.5	17.4540
1.5	1	7.58 (±3.8)	5	2.2137
	0.5	7.97 (±4.3)	35	2.3423
	0.2	8.96 (±4.1)	75	10.1730
	0.1	9.93 (±5.1)	82.5	0.2784
1	1	9.73 (±4.8)	15	3.0768
	0.5	10.81 (±6.0)	45	4.6470
	0.2	11.15 (±6.0)	82.5	0.0994
	0.1	11.81 (±5.6)	85	11.1124
0.5	1	17.46 (±10.5)	10	0.9718
	0.5	18.68 (±11.0)	42.5	3.6036
	0.2	19.49 (±11.4)	82.5	0.0202
	0.1	19.39 (±11.1)	95	0.0121
0.2	1	48.91 (±30.9)	12.5	0.9851
	0.5	53.96 (±35.6)	45	7.3513
	0.2	53.69 (±34.2)	82.5	11.0859
	0.1	58.39 (±36.7)	97.5	0.0080
0.1	1	135.44 (±91.4)	17.5	9.0054
	0.5	138.94 (±109.6)	45	8.3037
	0.2	150.26 (±117.5)	82.5	29.6101
	0.1	155.85 (±123.0)	95	38.6506

tion step improves the result and does not worsen the runtime that significantly. As expected however, with the increasing precision of the approximations, the found solution approaches the optimal one.

The difference in the increase of the runtimes is explained by the fact, that the approximation of the division is simulated by placement of additional constraints on the model, while the approximation of the objective function is handled by the Gurobi solver using the implemented piece-wise linear objective function.

Gurobi also provides the option to tweak certain parameters of the solver to improve the solving procedure. We experimented with the changing of priorities of certain model variables on which the algorithm branches and reducing or increasing various tolerances, but no significant improvement was observed. Finally, we also used the Gurobi Tuning Tool, which tries various parameter combinations on its own, to improve the runtime of the solver. This tool concluded that no further improvements can be made.

6.5 Branch-and-Price performance

The results of the test run of the Branch-and-Price algorithm are shown in the Table 4. We tested both the linear model and the enumeration model to solve the Pricing Problem. For each method we provide the mean runtime, mean number of expanded nodes and percentage of instances there were not solved in the limit. The mean runtime and mean number of expanded nodes are given with their respective standard deviations. The number of expanded nodes in the Branch-and-Price approach represents the number of nodes in the branching scheme tree. The number of nodes of the reference Branch-and-Bound is the total number of expanded nodes during the search.

Comparing the approaches to solve the Pricing Problem, we state that the linear model performed significantly better than the enumeration one. Although the enumeration model shows better runtime in several cases, such as $M = 4, N = 20$ and $M = 5, N = 20$, we note that the number of instances not solved in the time limit is significantly higher and therefore distorted the calculated results.

We see that the performance is generally worse compared to the reference Branch-and-Bound on instances where $M = 2$ a $M = 3$, with the exception of $M = 3, N = 20$ where the Branch-and-Price was able to solve more instances in the given time limit. For $M = 4, N = 14$ the results are already comparable to the reference algorithm. On the rest of the instances our algorithm outperforms the reference algorithm by a large margin. This is most likely due to the fact that the Branch-and-Price avoids symmetrical solutions during the computation and thus prunes the solution space significantly.

Table 4: Comparison of Branch-and-Price and the reference Branch-and-Bound.

		Branch-and-Price						Branch-and-Bound [Ranjbar *et al.*, 2012]		
		Linear Pricing model			Enumeration Pricing model					
machines	jobs	runtime [s]	nodes [–]	timeouts [%]	runtime [s]	nodes [–]	timeouts [%]	runtime [s]	nodes [–]	timeouts [%]
$M = 2$	$N = 14$	273.9 (±164.3)	1.1 (±0.2)	0	683.7 (±398.5)	1.1 (±0.4)	0	**0.2** (±0.0)	3.6K (±2.3K)	0
	$N = 16$	753.5 (±502.2)	1.2 (±0.7)	0	1.1K (±602.1)	1.3 (±0.7)	0	**0.7** (±0.1)	15.5K (±7.6K)	0
	$N = 18$	1.53K (±551.4)	1.3 (±0.7)	5	1.6K (±1.1K)	2.0 (±4.0)	10	**3.0** (±0.3)	31.5K (±32.4K)	0
	$N = 20$	2.32K (±485.2)	1.1 (±0.2)	20	1.6K (±676.8)	1.0 (±0.0)	50	**12.3** (±0.9)	159.0K (±133.7 K)	0
$M = 3$	$N = 14$	116.7 (±245.7)	6.4 (±14.5)	0	652.9 (±852.6)	11.1 (±18.7)	5	**6.0** (±1.1)	237.0K (±4.4K)	0
	$N = 16$	366.6 (±398.9)	11.3 (±16.9)	0	835.9 (±871.0)	9.4 (±20.2)	25	**60.3** (±11.0)	2.4M (±439.4K)	0
	$N = 18$	797.2 (±566.2)	10.6 (±21.4)	10	872.7 (±521.0)	7.1 (±8.5)	30	**507.6** (±76.8)	19.8M (±3.1)M	0
	$N = 20$	**1.7K** (±750.7)	12.6 (±20.4)	30	1.4K (±1.1K)	5.4 (±10.8)	40	3.4K (±0)	133.0M (±0)	95
$M = 4$	$N = 14$	26.8 (±18.4)	5.2 (±10.5)	0	312.1 (±352.1)	4.2 (±8.3)	5	**22.3** (±8.6)	253.2K (±192.6K)	0
	$N = 16$	**85.7** (±115.6)	9.4 (±14.3)	0	714.0 (±812.2)	7.8 (±17.7)	50	383.9 (±109.5)	5.7M (±3.2M)	0
	$N = 18$	**496.5** (±574.4)	20.0 (±20.1)	0	1.1K (±1.0K)	4.1 (±6.9)	45	–	–	100
	$N = 20$	**1.2K** (±893.9)	47.4 (±68.9)	5	926.0 (±481.1)	4.2 (±7.8)	70	–	–	100
$M = 5$	$N = 14$	**11.7** (±4.4)	1.5 (±2.0)	0	86.7 (±55.6)	1.1 (±0.5)	10	18.3 (±6.8)	518.0K (±192.2K)	0
	$N = 16$	**36.8** (±27.2)	10.1 (±13.6)	0	249.1 (±237.2)	4.0 (±6.7)	25	553.3 (±118.8)	15.6M (±3.4M)	0
	$N = 18$	**151.3** (±164.9)	26.0 (±33.0)	5	360.0 (±272.7)	2.9 (±5.7)	55	–	–	100
	$N = 20$	**1.2K** (±1.4)K	44.0 (±58.9)	10	646.2 (±384.2)	4.2 (±7.8)	70	–	–	100
$M = 6$	$N = 14$	10.9 (±4.6)	1.7 (±2.3)	0	183.5 (±533.3)	2.7 (±6.0)	0	**9.6** (±5.2)	100.1K (±101.6K)	0
	$N = 16$	**19.3** (±9.2)	3.3 (±4.9)	0	76.6 (±43.4)	1.4 (±1.3)	10	378.7 (±224.7)	4.7M (±4.8M)	0
	$N = 18$	**49.8** (±30.4)	9.6 (±13.8)	0	367.4 (±418.5)	7.4 (±9.8)	25	2.2K (±587.0)	19.2M (±26.8M)	90
	$N = 20$	**217.1** (±295.5)	54.0 (±88.0)	15	203.8 (±206.0)	1.0 (±0.0)	50	–	–	100

6.6 Two-machines model performance

Table 5 shows the mean running time and its standard deviation of the Two-machines model compared to the Branch-and-Price and the reference Branch-and-Bound algorithms for instances with 2 machines. We give results for the Two-machines model run restricted to a single CPU core and to four CPU cores. Clearly, the Two-machines model outperforms

the Branch-and-Price and is comparable to the Branch-and-Bound on the instances with smaller number of jobs, but shows significantly better scaling properties as the number of jobs increases. We also note that both the Branch-and-Price and the reference Branch-and-Bound algorithms were unable to solve the instances with $N \geq 40$ within given time limit.

Table 5: Comparison of Two-machines model, Branch-and-Price and the reference Branch-and-Bound.

machines	jobs	Two-machines model (1 core) runtime [s]	Two-machines model (4 cores) runtime [s]	Branch-and-Price runtime [s]	Branch-and-Bound [Ranjbar *et al.*, 2012] runtime [s]
$M = 2$	$N = 14$	1.1 (±0.8)	0.3 (±0.2)	273.9 (±164.3)	**0.2** (±0.0)
	$N = 16$	1.0 (±0.9)	**0.3** (±0.3)	753.5 (±502.2)	0.7 (±0.1)
	$N = 18$	1.3 (±1.2)	**0.4** (±0.3)	1.53K (±551.4)	3.0 (±0.3)
	$N = 20$	1.5 (±1.1)	**0.4** (±0.3)	2.32K (±485.2)	12.3 (±0.9)
	$N = 40$	2.5 (±1.9)	**0.7** (±0.6)	–	–
	$N = 80$	5.4 (±4.2)	**1.5** (±1.2)	–	–
	$N = 200$	14.6 (±8.9)	**3.9** (±2.4)	–	–
	$N = 500$	65.1 (±37.3)	**16.9** (±9.7)	–	–

Further, Figure 8 shows the mean running time of the Two-machines model run on instances with 14, 16, 18, 20, 40, 80, 200 and 500 jobs, with varying number of allocated CPU cores, namely 1, 4, 8 and 16 cores. Even for the largest instances ($N = 500$) we were able find the solution in about a minute for a single core run and in less than 20 seconds for the increased number of allocated cores. The model shows excellent parallel scaling ability.

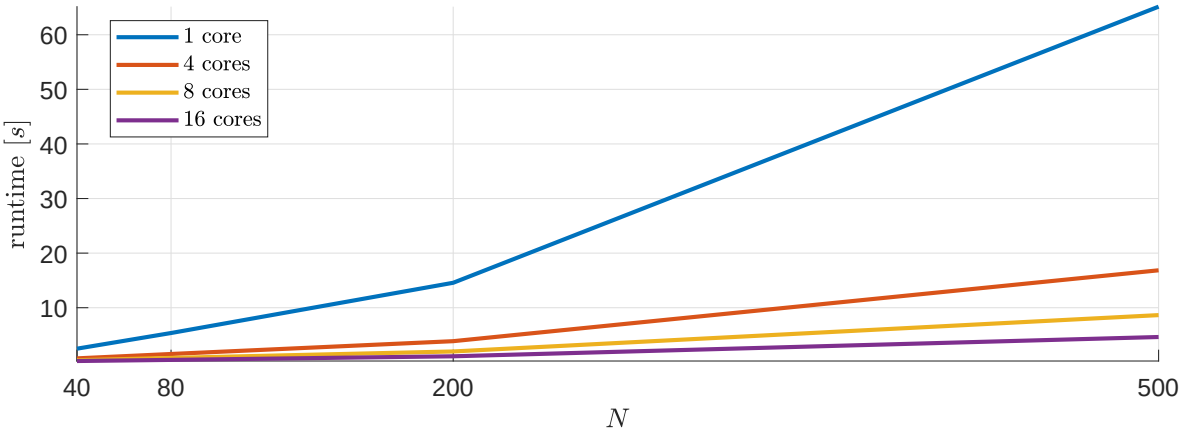


Figure 8: Mean running times of Two-machines model

6.6.1 ξ evaluation

Figure 9 shows the histogram of calculated ξ values from the optimal solution of the instances. It holds in all the cases that $\xi \geq 0.8$ and therefore we claim that the performed approximation of Equation (5.22) is justified.

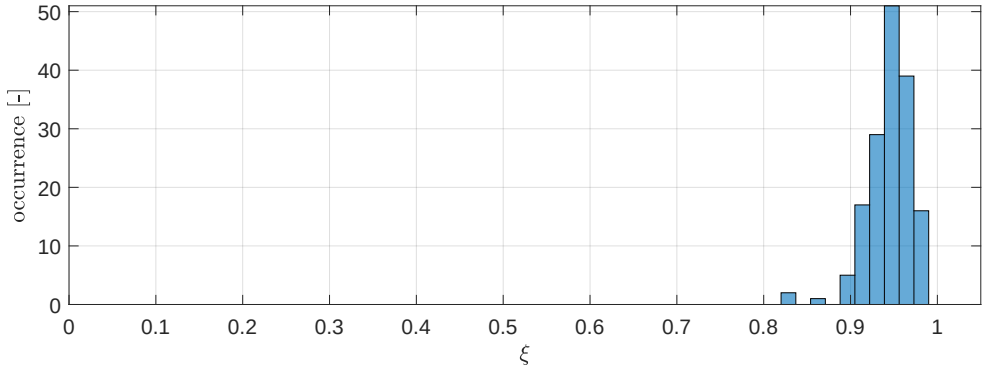


Figure 9: Histogram of ξ values calculated from the optimal solution to instances.

7 Conclusion

This thesis focuses on a stochastic scheduling problem, where the processing time of each job is not exactly known and is given by normal distribution. This problem is formulated as a β-robust scheduling problem where the objective is to maximize the probability the schedule is completed before a given common due date.

One of the most significant contributions of this work is the Branch-and-Price decomposition of the problem. In most cases, the performance is significantly better then the Branch-and-Bound algorithm proposed by [Ranjbar *et al.*, 2012]. It is also easier to implement and does not produce symmetrical solutions, which prunes the search state space greatly. On most of the instances with 3 machines, the Branch-and-Price method lacks in performance. However, we note that on the largest 3 machine instances ($M = 3, N = 20$) we outperform the reference Branch-and-Bound and a better performance can be expected on larger instances as well. On instances with more than 3 machines, the proposed Branch-and-Bound shows significantly better scaling properties and outperforms the reference algorithm by a large margin.

A notable difference is observed on instances with exactly 2 machines. An examination had shown, that on these instances the Pricing Problem requires more time to be solved and more patterns are generated as well. Therefore, we devised a Two-machines model that solves this special case. We show, that this model is able to solve instances with up to 500 jobs in several seconds as compared to the reference Branch-and-Bound which cannot solve instances with 40 jobs and more in the given time limit. This approach dominates the reference algorithm and Branch-and-Price algorithm in every case.

Finally, we also provide a linear model of the problem. We note that its performance as a standalone solver is not satisfactory. However, we reuse it to solve the Pricing Problem and get a better results as compared to the Variance Enumeration model.

We note that the main limitation of the problem is its restriction to normal distribution of the processing times. However, the proposed approach can be extended to any distribution that belong to the family of *closed under convolution* distributions.

We conclude with the remark, that part of the results, regarding the Branch-and-Price decomposition, were submitted and accepted for the International Joint Conferences on Artificial Intelligence 2019 (IJCAI-2019).

References

Igal Adiri, John Bruno, Esther Frostig, and AHG Rinnooy Kan. Single machine flow-time scheduling with a single breakdown. *Acta Informatica*, 26(7):679–696, 1989.

S Alimoradi, M Hematian, and Ghasem Moslehi. Robust scheduling of parallel machines considering total flow time. *Computers & Industrial Engineering*, 93:152–161, 2016.

Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.

Zhiqi Chang, Jian-Ya Ding, and Shiji Song. Distributionally robust scheduling on parallel machines under moment uncertainty. *European Journal of Operational Research*, 272(3):832–846, 2019.

Abraham Charnes and William W Cooper. Programming with linear fractional functionals. *Naval Research logistics quarterly*, 9(3-4):181–186, 1962.

TCE Cheng and CCS Sin. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47(3):271–292, 1990.

Richard L Daniels and Janice E Carrillo. β-robust scheduling for single-machine systems with uncertain processing times. *IIE transactions*, 29(11):977–985, 1997.

Richard L Daniels and Panagiotis Kouvelis. Robust scheduling to hedge against processing time uncertainty in single-stage production. *Management Science*, 41(2):363–376, 1995.

Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2006.

Didier Dubois, Helene Fargier, and Philippe Fortemps. Fuzzy scheduling: Modelling flexible constraints vs. coping with incomplete knowledge. *European Journal of Operational Research*, 147(2):231–252, 2003.

Hesham El-Rewini and Ted G. Lewis. Scheduling parallel program tasks onto arbitrary target machines. *Journal of parallel and Distributed Computing*, 9(2):138–153, 1990.

Andreas T Ernst, Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. Staff scheduling and rostering: A review of applications, methods and models. *European journal of operational research*, 153(1):3–27, 2004.

R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In P.L. Hammer, E.L. Johnson, and B.H. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287 – 326. Elsevier, 1979.

Panos Kouvelis, Richard L Daniels, and George Vairaktarakis. Robust scheduling of a two-machine flow shop with uncertain processing times. *Iie Transactions*, 32(5):421–432, 2000.

Chung-Cheng Lu, Shih-Wei Lin, and Kuo-Ching Ying. Robust scheduling on a single machine to minimize total flow time. *Computers & Operations Research*, 39(7):1682–1691, 2012.

Bart L Maccarthy and Jiyin Liu. Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *The International Journal of Production Research*, 31(1):59–79, 1993.

István Módos, Přemysl Šůcha, and Zdeněk Hanzálek. Algorithms for robust production scheduling with energy consumption limits. *Computers & Industrial Engineering*, 112:391–408, 2017.

Michael Pinedo. Stochastic scheduling with release dates and due dates. *Operations Research*, 31(3):559–572, 1983.

Michael Pinedo. *Scheduling*. Springer, 2012.

Salih Ramazan and Roussos Dimitrakopoulos. Production scheduling with uncertain supply: a new solution to the open pit mining problem. *Optimization and Engineering*, 14(2):361–380, 2013.

Mohammad Ranjbar, Morteza Davari, and Roel Leus. Two branch-and-bound algorithms for the robust parallel machine scheduling problem. *Computers & Operations Research*, 39(7):1652 – 1660, 2012.

Ihsan Sabuncuoglu and M Bayız. Analysis of reactive scheduling problems in a job shop environment. *European Journal of operational research*, 126(3):567–586, 2000.

Stephen F Smith. Reactive scheduling systems. In *Intelligent scheduling systems*, pages 155–192. Springer, 1995.

Pieter S Stepaniak, Christiaan Heij, Guido HH Mannaerts, Marcel de Quelerij, and Guus de Vries. Modeling procedure and surgical times for current procedural terminology-anesthesia-surgeon combinations and evaluation in terms of case-duration prediction and operating room efficiency: a multicenter study. *Anesthesia & Analgesia*, 109(4):1232–1245, 2009.

Paul PM Stoop and Vincent CS Wiers. The complexity of scheduling in practice. *International Journal of Operations & Production Management*, 16(10):37–53, 1996.

Richard Štec, Antonín Novák, Přemysl Šůcha, and Zdeněk Hanzálek. Scheduling jobs with stochastic processing time on parallel identical machines. In *Proceedings of IJCAI 2019*. International Joint Conferences on Artificial Intelligence, to appear 2019.

Juite Wang. A fuzzy robust scheduling approach for product development projects. *European Journal of Operational Research*, 152(1):180–194, 2004.