A STOCHASTIC PROGRAMMING APPROACH TO THE SINGLE MACHINE MAKESPAN PROBLEM WITH RANDOM BREAKDOWNS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

TARIK GÜREL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

DECEMBER 2022

Approval of the thesis:

A STOCHASTIC PROGRAMMING APPROACH TO THE SINGLE MACHINE MAKESPAN PROBLEM WITH RANDOM BREAKDOWNS

submitted by **TARIK GÜREL** in partial fulfillment of the requirements for the degree of **Master of Science** i**n Industrial Engineering, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Esra Karasakal
Head of the Department, **Industrial Engineering** _____

Prof. Dr. Meral Azizoğlu
Supervisor, **Industrial Engineering, METU** _____

Assist. Prof. Dr. Sakine Batun
Co-Supervisor, **Industrial Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Serhan Duran
Industrial Engineering, METU _____

Prof. Dr. Meral Azizoğlu
Industrial Engineering, METU _____

Assist. Prof. Dr. Sakine Batun
Industrial Engineering, METU _____

Assoc. Prof. Dr. Melih Çelik
Business Administration, University of Bath. _____

Assoc. Prof. Dr. Özlem Karsu
Industrial Engineering, Bilkent University _____

Date: 02.12.2022

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name Last name : Tarık Gürel

Signature :

ABSTRACT

A STOCHASTIC PROGRAMMING APPROACH TO THE SINGLE MACHINE MAKESPAN PROBLEM WITH RANDOM BREAKDOWNS

Gürel, Tarık
Master of Science, Industrial Engineering
Supervisor : Prof. Dr. Meral Azizoğlu
Co-Supervisor: Assist. Prof. Dr. Sakine Batun

December 2022, 67 pages

In this thesis, we consider a single machine scheduling problem with random breakdowns. There is a single breakdown whose occurrence times follow a discrete distribution with known probabilities. We aim to minimize the expected makespan and propose a stochastic programming approach.

We propose two stage stochastic programming models and a branch and bound algorithm. We enhance the performance of the branch and bound algorithm with an efficient branching scheme and powerful lower bounds.

The results of our computational experiments have shown that the stochastic programming models can solve small-sized instances and the branch and bound algorithm is capable of solving medium sized instances in reasonable times.

Keywords: Single Machine Sequencing, Makespan, Stochastic Programming Models, Branch and Bound Algorithm, Random Breakdown

ÖZ

STOKASTİK PROGRAMLAMA YAKLAŞIMI İLE RASSAL ARIZA ETKİSİNDE TEK MAKİNE ÇİZELGELEME

Gürel, Tarık
Yüksek Lisans, Endüstri Mühendisliği
Tez Yöneticisi: Prof. Dr. Meral Azizoğlu
Ortak Tez Yöneticisi: Dr. Öğr. Üyesi Sakine Batun

Aralık 2022, 67 sayfa

Bu tezde, rassal arıza varlığında tek makinenin çizelgeleme problemini ele almaktayız. Gerçekleşme zamanı bilinen olasılık altında ayrık dağılım özelliği gösteren bir arıza mevcuttur. Beklenen son işin tamamlanma süresini azaltmayı amaçlamaktayız ve bunun için stokastik programlamı yaklaşımı önermekteyiz.

Belirtilen amaç doğrultusunda iki seviyeli stokastik programlama modelleri ve dal/sınır algoritması önermekteyiz. Etkili dallanma planı ve güçlü alt sınırlar ile dal/sınır algoritmasının performansını arttırmaktayız.

Hesaplamalarımızın sonuçları stokastik programlama modellerimizn küçük ölçekli problemlerde, dal-sınır algoritmasının ise orta ve büyük ölçekli problemlerde makul sürelerde çalıştığını göstermiştir.

Anahtar Kelimeler: Tek Makine Sıralaması, Son İş Tamamlanma Süresi, Stokastik Programlama Modelleri, Dal-Sınır Algoritması, Rassal Arıza

To my family and friends…

ACKNOWLEDGMENTS

Firstly, I would like to express my deepest gratitude to my supervisor Prof. Dr. Meral Azizoğlu and co-supervisor Assist. Prof. Dr. Sakine Batun for their invaluable assistance, encouragement, patience and endless support. I am extremely grateful and happy to work together.

I would like to thank jury members for their valuable comments on the thesis and positive attitudes during my thesis defense.

I would like to present my special thanks to my family. I am indebted to my family's unconditional and loving support. They are always with me throughout not only this thesis but also my whole life.

I am also thankful to my company, ASELSAN, for its support during my study.

Lastly, very special thanks to my friends. They always support and motivate me during my journey.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vi

ACKNOWLEDGMENTS viii

TABLE OF CONTENTS ix

LIST OF TABLES xi

LIST OF FIGURES xiv

CHAPTERS

1 INTRODUCTION 1

2 LITERATURE REVIEW 5

2.1 Maintenance Studies 5

2.1.1 Deterministic Maintenance 6

2.1.2 Stochastic Maintenance 7

2.2 Stochastic Programming for Machine Scheduling Problems 8

3 PROBLEM DEFINITION AND FORMULATION 11

3.1 Problem Description 11

3.2 Stochastic Programming (SP) Basics 12

3.3 Stochastic Programming Models 13

3.3.1 Precedence Based Mixed Integer Linear Program 13

3.3.2 Position Based Mixed Integer Linear Program 16

3.4 An Example 17

3.5 Expected Value Problem and the Value of the Stochastic Solution 19

3.6 An Example 21

4 BRANCH & BOUND METHOD 25
4.1 Basics of Branch and Bound Method 25
4.2 Proposed Branch and Bound Algorithm 26
4.3 Bounding Mechanisms 28
5 COMPUTATIONAL EXPERIMENTS 37
5.1 Data Generation 37
5.2 Performance Measures 39
5.3 Analysis of the Results 39
5.3.1 Stochastic Programming 40
5.3.2 Branch & Bound Algorithm 42
5.3.3 Expected Value Problem and the Value of the Stochastic Solution 59
6 CONCLUSION 63
REFERENCES 65

LIST OF TABLES

TABLES

Table 3.1 Processing times of jobs in example 17

Table 3.2 Breakdowns times and probabilities of an example 17

Table 3.3 Proposed schedule that shows the breakdown of scenario 1 18

Table 3.4 Proposed schedule that shows the breakdown of scenario 2 18

Table 3.5 Proposed schedule that shows the breakdown of scenario 3 19

Table 3.6 Output of the knapsack problem 21

Table 3.7 Schedule of EV solution for scenario 1 22

Table 3.8 Schedule of EV solution for scenario 2 22

Table 3.9 Schedule of EV solution for scenario 3 22

Table 4.1 Processing time data 27

Table 4.2 Breakdown time data 27

Table 4.3 Processing time of jobs in example 29

Table 4.4 Breakdown time and occurrence probability data 29

Table 4.5 Processing time data for 7-job, 6-scenario example instance 33

Table 4.6 Breakdown time and occurrence probability data for 7-job, 6-scenario example instance 33

Table 5.1 Performances of *SP1* and *SP2*, $p_i \sim U[1,10]$, n=10..... 40

Table 5.2 Performances of *SP1* and *SP2*, $p_i \sim U[1,100]$, n=10..……..... 40

Table 5.3 CPU Times of SP1, $p_i \sim U[1,100]$..... 41

Table 5.4 Computational Results for the Branch and Bound Algorithm for Set 1, Plan 1 and $p_i \sim U[1,10]$ ……...42

Table 5.5 Computational Results for the Branch and Bound Algorithm for Set 1, Plan 2 and $p_i \sim U[1,10]$ ……...42

Table 5.6 Computational Results for the Branch and Bound Algorithm for Set 2, Plan 1 and $p_i \sim U[1,10]$ ……...43

Table 5.7 Computational Results for the Branch and Bound Algorithm for Set 2, Plan 2 and $p_i \sim U[1,10]$ ……...43

Table 5.8 Computational Results for the Branch and Bound Algorithm for Set 1, Plan 1 and $p_i \sim U[1,100]$ ……...43

Table 5.9 Computational Results for the Branch and Bound Algorithm for Set 1, Plan 2 and $p_i \sim U[1,100]$ ……...44

Table 5.10 Computational Results for the Branch and Bound Algorithm for Set 2, Plan 1 and $p_i \sim U[1,100]$ ……...44

Table 5.11 Computational Results for the Branch and Bound Algorithm for Set 2, Plan 2 and $p_i \sim U[1,100]$ ……...44

Table 5.12 Computational Results for the Branch and Bound Algorithm for Set 1, Plan 1, higher *n* values and $p_i \sim U[1,100]$ …….....……46

Table 5.13 Computational Results for the Branch and Bound Algorithm for Set 1, Plan 2, higher *n* values and $p_i \sim U[1,100]$ …….....……46

Table 5.14 Computational Results for the Branch and Bound Algorithm for Set 2, Plan 1, higher *n* values and $p_i \sim U[1,100]$ …….....……46

Table 5.15 Computational Results for the Branch and Bound Algorithm for Set 2, Plan 2, higher *n* values and $p_i \sim U[1,100]$ …….....……47

Table 5.16 Distribution of unsolved instances to specific problem settings 51

Table 5.17 Results of Performance Measure when LBs are not used for *n*=10 53

Table 5.18 Results of Performance Measure when LBs are used for *n*=10 53

Table 5.19 The number of unsolved instances when $p_i \sim U[1,10]$ 54

Table 5.20 The CPU times and number of nodes when $p_i \sim U[1,10]$..... 55

Table 5.21 The number of unsolved instances when $p_i \sim U[1,100]$ 56

Table 5.22 The CPU times and number of nodes when $p_i \sim U[1,100]$..... 56

Table 5.23 Number of instances (first feasible solution = optimal solution) for different problem settings 58

Table 5.24 Deviations for instances that first feasible solution is not equal to optimal solution 58

Table 5.25 EEV-SP values when $p_i \sim U[1,100]$ for n=10 59

Table 5.26 VSS values for *n*=50, *m*=5 and $p_i \sim U[1,100]$ 60

Table 5.27 VSS values for *n*=100, *m*=5 and $p_i \sim U[1,100]$..... 60

Table 5.28 VSS values for *n*=50, *m*=3 and $p_i \sim U[1,100]$..... 61

Table 5.29 VSS values for *n*=50, *m*=5 and $p_i \sim U[1,10]$..... 61

LIST OF FIGURES

FIGURES

Figure 4.1 A partial branching tree.....26

CHAPTER 1

INTRODUCTION

In this study, we consider a single machine makespan problem with uncertain breakdowns. A single machine is an important concern for many manufacturers who have to operate a single prestigious machine like a robot or a CNC machine. Moreover, a single machine may represent a collection of many machines or a bottleneck machine.

The maximum flow time overall jobs, so called makespan, is an important concern of manufacturing, in particular when all jobs in a lot should wait for each other to be moved from the operation shop floor. The breakdowns decrease the efficiency of the manufacturing operations through manufacturing cost increases, quality deterioration, and makespan increases.

The scheduling theory mostly assumes that the resources are continuously available to process the operation and there are no breakdowns. This assumption may contradict the practical applications where the machines are to be maintained at some defined times once they break down. A former situation is a deterministic event whereas the latter case is a random event that occurs in stochastic environments.

Random breakdowns are relatively less studied in stochastic scheduling literature. The majority of the literature on stochastic scheduling assumes that uncertainty arises due to the processing times. The studies which deal with breakdowns assume that the breakdowns are governed by known probability distribution functions and propose optimal policies to minimize the expected performance measure.

Recognizing this gap in the literature, we consider a single breakdown whose occurrence time follows a discrete distribution with known probabilities. During the short planning horizon of a scheduling environment, it is realistic to make such an assumption once a machine breaks down and thereafter maintained, it will not re-break down till the completion of the last job. We call any occurrence event a scenario and assume that there are limited scenarios. Each scenario that is defined by a time point may be representative of the time interval that it is in. We aim to minimize the expected makespan which is the collection of all makespan values weighted by their probabilities.

We propose two stochastic programming models one of which is precedence based and the other is position based. The models are mixed integer linear programs with a lot of binary variables, hence their applications are limited to small-sized problem instances. For medium to large-sized problem instances, we propose a branch and bound algorithm whose efficiency is enhanced by an efficient branching scheme and powerful lower bounding mechanisms.

To sum up, we contribute to the existing literature in the following two ways:

i. proposing the first stochastic programming model for a scheduling problem with random breakdowns.

ii. proposing the first optimization approach (branch and bound algorithm) to a single machine stochastic programming model

The rest of the thesis is organized as follows. In Chapter 2, we review the literature on deterministic problems with scheduled breakdowns and stochastic problems with uncertain parameters. In addition, we discuss the stochastic programming approaches for machine scheduling problems. Chapter 3 defines the problem, gives the stochastic programming models, and presents the models used to find the value of the stochastic solution. In Chapter 4, we present the branch and bound algorithm along with the mechanisms used to enhance its efficiency. Chapter 5 discusses our

computational experiments, and we conclude the study in Chapter 6, where the main conclusions and future research directions are stated.

CHAPTER 2

LITERATURE REVIEW

Throughout the years, there have been many studies about scheduling with preventive maintenance all over the world and one of the reasons why it can be such a popular study area is that lots of variants as study topics can be derived by changing the problem environment. Number of machines, scheduling methods, maintenance strategies and objectives are examples of what the problem environment is made up of.

The majority of the studies carried out in the earlier years is related to deterministic problems. Data corresponding to processing time, breakdown time and so on are already known, and solution can be found by using the data from existing algorithms and solution methods. However, this situation does not reflect real life cases. Particularly in the manufacturing sector, companies regularly face maintenance activities (corrective maintenance) performed after random breakdowns. These events have played a vital role in the emerging of studies focused on stochastic problems.

We first review the maintenance literature and then the stochastic programming studies for all scheduling environments.

2.1 Maintenance Studies

We review the maintenance literature under two headings: deterministic and stochastic maintenance

2.1.1 Deterministic Maintenance

Chen (2009) carried out study about a single-machine scheduling problem with periodic maintenance where objective is minimizing number of tardy jobs. There were several maintenance periods in which an amount of time was required for the performance. In the lights of these information, Chen (2009) studied how to schedule for jobs and maintenance period. A heuristic algorithm based on Moore-Hodgson's algorithm was developed to obtain near-optimal schedule. To decide whether the developed heuristic well performed or not, branch and bound method also implemented and comparison was made. Liu et al. (2015) addressed the same problem that has been studied by Chen (2009). They suggested an improved branch and bound algorithm that involves new effective dominance rules and strong lower bounds. Low et al. (2010) dealt with single machine scheduling problem under the deterministic environment aiming to minimize the makespan. Machine is not available at all time due to periodic maintenance activities scheduled after a periodic time interval. They focused on heuristic methods to solve NP-hard problem, and decreasing order with first fit, one of the proposed heuristics, performed well according to computational results. Chen et al. (2020) also considered single machine scheduling problem with preventative maintenance to minimize the makespan in which jobs were non-resumable, and maintenance intervals were flexible. They tried to solve this problem by utilizing from mixed-integer programming models (MIP) and branch and bound method (B&B) initially. They concluded that these methods were satisfactory for the small size problems, and B&B heavily dominated the MIP. Then, four different heuristics were designed to reach near-optimal solution quickly for the large size problem. The result of their experiments revealed the satisfactory performance of heuristics, especially Minimum Waste and Lower Bound Index. Batun and Azizoğlu (2009) worked on total flow time single machine problem with preventative maintenance. In this problem, the assumption is that jobs must be restarted when disrupted by the

maintenance activities. Another assumption is that starting time and duration of these activities are already known. Under these conditions, they proposed Branch and Bound as a solution method. Their computations showed that Branch and Bound method performed well in a large sized problem, up to 80 jobs.

Hariga (1994) considered the maintenance problem with m non-identical machines. The objective is to determine cyclic overhaul schedule for a manufacturing system. Hariga (1994) also made some assumptions according to the impact of maintenances (overhauls). While major overhauls bring the machines to new conditions (operating age becomes zero), minor overhauls only restore machines to specified operating conditions. Hariga (1994) was developed a heuristic algorithm to determine overhaul schedule instead of optimization model since solving the latter was more difficult. Also, the computations yielded good results for the heuristic algorithm.

2.1.2 Stochastic Maintenance

Cassady and Kutanoglu (2005) made a study about single machine scheduling problem by incorporating preventative maintenance to minimize the total expected weighted completion time. They considered that machines are subject to failures, and the failure rate is determined by using the Weibull distribution. They proposed an optimization model to solve this type of problem, and they compared the performance of integrated scheduling with non-integrated scheduling. Pan et al. (2010) considered similar problem settings to Cassady and Kutanoglu (2005), but their objective was minimizing the maximum weighted tardiness. They proposed an integrated scheduling model that includes production scheduling and preventive maintenance to deal with this type of problem. Wang and Liu (2013) investigated single machine production scheduling with preventative maintenance planning, where objective is minimizing total weighted completion time of jobs. Machine

breakdowns occur, and the time to breakdown is subject to a Weibull probability distribution. Branch and bound was used as solution approach whose efficiency is enhanced with powerful upper and lower bounds. After these efforts, they solved problems with up to 18 jobs in a reasonable time. Halim et al. (2020) proposed a hybrid method that is a combination of a genetic algorithm with a Monte Carlo simulation method for solving single machine scheduling with random machine breakdowns where the objective is integrating the production and preventative maintenance to minimize the total completion time. Firstly, they utilized simulation to predict the breakdown time, and the genetic algorithm was used through easy to implement for scheduling the jobs.

2.2 Stochastic Programming for Machine Scheduling Problems

van den Akker et al. (2018) considered a stochastic single machine environment where changes in processing times of jobs are due to disturbance. Their approach was to propose recovery actions to be implemented to reach optimal solution for minimizing number of late jobs when a change occurs. To achieve that, they proposed to use a combination of two stage stochastic programming and recoverable robustness. Several algorithms, dynamic programming, branch and bound and branch and price, were used as solution methods for this problem. After computations, they concluded that dynamic programming was the worst method, and branching algorithms were better. In spite of the fact that there were no difference between branching methods, they proposed using branch and bound as it was easily implemented.

Khamis and M'Hallal (2011) also utilized two stage stochastic programming, but problem environment was totally different. They dealt with the parallel machine scheduling problem in which due dates of jobs are uncertain, and their objective was to determine optimal machine capacities to maximize profit. They divided the model into two stages, the first stage and the second stage. While they determined

optimal capacities for each machine in the first stage, they tried to estimate profit with respect to determined capacities in the second stage. To solve this problem, they combined ranking and selection procedure and used sample average approximation. Also, they used branch and bound method to solve the second stage of the problem.

Özçelik et al. (2021) considered single machine scheduling problem with stochastic sequence dependent set-up times, where objective function is minimizing total expected cost that consists of set-up, tardiness and earliness cost. They develop a mathematical model that includes a combination of single machine scheduling with stochastic sequence-dependent setup time and cost function, however, could not find the optimal solution in CPLEX in a reasonable time. For this reason, they looked for heuristic and metaheuristic methods, and decided to use harmony search algorithm due to its great search ability and simplicity. Then, they proposed two stage stochastic programming model with harmony search heuristic to solve the large-sized problems.

Atakan et al. (2016) studied value-at-risk minimization in single machine scheduling problems under unforeseeable conditions and parameters. They benefited from Lagrangian relaxation-based scenario decomposition method for obtaining lower bound, and proposed risk-averse stochastic programming model.

van den Akker and Hoogeven (2008) considered single machine scheduling problems with stochastic processing times, to minimize the number of late jobs. They used a chance constraint to decide whether job is stochastically completed on time or late. A theorem was developed for different stochastic processing times distributions to convert stochastic problems to deterministic problems. They also scrutinized maximizing expected number of on time jobs.

CHAPTER 3

PROBLEM DEFINITION AND FORMULATION

In this chapter, we define our problem, give the basics of the stochastic programming approach and present two alternate stochastic programming models for the defined problem. We also report on the model used to assess the value of our stochastic programming approach.

3.1 Problem Description

We consider a single machine scheduling model with n jobs. Task i has deterministic processing time of p_i time units and the machine is subject to a single breakdown whose occurrence time is probabilistic. The jobs are indexed in non-decreasing order of processing times: if $j > i$, then $p_j \geq p_i$.

There are m possible times of the breakdown each of which is referred to as a scenario. Scenario k is characterized by its breakdown time s_k, breakdown duration d_k and occurrence probability π_k.

We make the following additional assumptions:

- All jobs are available at time zero and most one job can be processed at a time.
- The jobs are non-resumable, i.e., they are restarted once interrupted by the breakdown.
- The uncertainty in machine breakdown and repair durations is represented by a set of scenarios.

- The breakdown time for each scenario cannot be greater than the sum of jobs' processing time.

Our problem is to sequence the n jobs on a single machine to minimize the expected completion time of the last sequenced task, i.e., expected makespan. We express the expected makespan as $\sum_{k=1}^{m} \pi_k * C_{max,k}$ where $C_{max,k}$ is the completion time of the last job according to scenario k. Makespan is an important concern in batch manufacturing systems where the jobs arrive in lots and the next batch start after all jobs of the current batch is completed.

3.2 Stochastic Programming (SP) Basics

Stochastic programming is a mathematical program which includes problem parameters that are not known with certainty. Real world problems almost always involve stochastic parameters instead of fixed parameters known in advance. In stochastic programming, these parameters are estimated by using probability distributions.

Two-stage stochastic programming (SP) framework is the simplest structure stochastic programming where the decision variables are divided into two groups as the first-stage and the second-stage decision variables. The first-stage decisions ($\boldsymbol{x}$) are made and implemented before the resolution of the uncertainty, whereas the implementation of the second-stage decision variables ($\boldsymbol{y}$) takes place after the resolution of the uncertainty, i.e., after the second-stage parameters (random vector $\boldsymbol{\xi}$) become known.

Aligned with this structure, Birge (2011) states extensive form of two stage SP model as follows:

$$\min c^T x + E_\xi Q(x, \xi)$$

s.t

$$Ax = b$$

$$x \geq 0$$

where $Q(x, \xi)$ = min $\{q^T y \mid Wy = h - Tx, y \geq 0\}$, and ξ is the vector composed by the components of q^T , h^T, and T. In addition, E_ξ denote mathematical expectation with respect to ξ. Also, W is assumed as fixed.

3.3 Stochastic Programming Models

In this section, we present two SP models that are precedence based and position based.

3.3.1 Precedence Based Mixed Integer Linear Program

The model forms the solution by using the precedence sequences of the jobs in the optimal solution. Hence we define our decision variable as an indicator variable that takes value 1 if a job precedes another task in the optimal solution. We hereafter refer to this model as SP_1.

SP_1 uses the following notation:

Indices

i, j= task (job) indices, 1,…,n

k= scenario index, 1,…,m

Parameters

p_i = processing time of job i i =1,…,n

π_k = occurrence probability of scenario *k* k=1,…,m

s_k = breakdown time of scenario *k* k=1,…,m

M_{1k} = an upper bound on the completion time in scenario *k* k=1,…,m

M_{2k} = an upper bound on the completion time in scenario *k* k=1,…,m

M = an upper bound on the subtour constraint

d_k = breakdown duration in scenario k k=1,…,m

Decision Variables

$x_{ij} = \begin{cases} 1 \text{ if job i precedes job } j \text{ (general precedence, not immediate precedence)} \\ 0 \text{ otherwise} \end{cases}$

where *i, j=1,…,n*

$y_{jk} = \begin{cases} 1 \text{ if job } j \text{ is completed before breakdown in scenario } k \\ 0 \text{ otherwise} \end{cases}$

where *j=1,…,n* and *k=1,…,m*

$C_{max,k}$ = makespan value in scenario *k* where *k*=1,…,*m*

C_j = *d*ummy completion time of job *j* where *j=1,…,n*

Mathematical Model

Objective Function: $\min \sum_k \pi_k * C_{max,k}$ (1)

Subject to

$x_{ij} + x_{ji} = 1$ $\forall j > i$ (2)

$\sum_{i \neq j} p_i x_{ij} + p_j \leq s_k + M_{1k}(1 - y_{jk})$ $\forall j, k$ (3)

where $M_{1k} = \sum_j (p_j) - s_k$

$\sum_{i \neq j} p_i x_{ij} + p_j \geq s_k - M_{2k}(y_{jk})$ $\forall j, k$ (4)

where $M_{2k} = s_k$

$$C_{max,k} = (s_k + d_k) + \sum_j p_j(1 - y_{jk}) \qquad \forall k: s_k < \sum_j p_j \qquad (5)$$

$$C_j \geq C_i + p_j - M(1 - x_{ij}) \qquad \forall j \neq i \qquad (6)$$

where $M = 2 \sum_j p_j + max_k (d_k)$

$$x_{ij} \in \{0,1\} \qquad \forall j \neq i \qquad (7)$$

$$y_{jk} \in \{0,1\} \qquad \forall j,k \qquad (8)$$

$$C_j \geq 0 \qquad \forall j \qquad (9)$$

$$C_{max,k} \geq 0 \qquad \forall k \qquad (10)$$

The objective function (1) corresponds to minimizing the expected makespan. The makespan values are calculated for each scenario separately, then, the values are multiplied by the occurrence probability of the scenario, π_k, and summed up.

Constraint set (2) ensures that either i precedes j or j precedes i, thus only one x variable for the same i and j values is equal to one. Constraint sets (3) and (4) are formulated to make sure that $y_{jk} = 1$ if j is completed before s_k and $y_{jk} = 0$ if j is completed after s_k, respectively. Constraint set (5) computes the makespan for each scenario. This constraint is the sum of breakdown time, breakdown duration and the set of jobs that have been operated after the breakdown. Constraint (6) can be considered as subsequence elimination constraints. It makes sure that job j starts after job i completes if $x_{ij} = 1$. Note that C_j variables do not represent the real completion times. Constraint sets (7) and (8) are the binary set restrictions on the variables. Constraint sets (9) and (10) ensure that variables take nonnegative values.

3.3.2 Position Based Mixed Integer Linear Program

The model forms the solution by using the assignments of the jobs to the positions. Hence we define our decision variable as an indicator variable that takes value 1 if a job is assigned to a particular position. We keep the position assignments identical over all scenarios. We hereafter refer to the model as SP_2.

SP_2 uses the same parameters and indices with SP_1. Additionally, we define an index r such that: $r = \begin{cases} 1 & \text{before breakdown} \\ 2 & \text{after breakdown} \end{cases}$

Decision Variables

x_{jtrk}

$= \begin{cases} 1 \text{ if job } j \text{ is assigned to position } t \text{ and completed } \begin{cases} \text{before } (r=1) \\ \text{after } (r=2) \end{cases} \text{ the breakdown in scenario } k \\ 0 \text{ otherwise} \end{cases}$

where j=1,…,n, t=1,…,n, r=1,2 and k=1,…,m.

$C_{max,k} =$ makespan value under scenario k where k=1,…,m.

Mathematical Model

Objective Function: $\min \sum_k \pi_k * C_{max,k}$ (11)

Subject to

$\sum_{t,r} x_{jtrk} = 1$ $\forall j,k$ (12)

$\sum_{j,r} x_{jtrk} = 1$ $\forall t,k$ (13)

$\sum_{j,r} j * x_{jtrk} = \sum_{j,r} j * x_{jtr(k+1)}$ $\forall t, k < m$ (14)

$\sum_{j,t} p_j x_{jt1k} \leq s_k$ $\forall k$ (15)

$\sum_j x_{j(t+1)2k} \geq \sum_j x_{jt2k}$ $\forall k, t < n$ (16)

$C_{max,k} = s_k + d_k + \sum_{j,t} p_j x_{jt2k}$ $\forall k: s_k < \sum_j p_j$ (17)

$x_{jtrk} \in \{0,1\}$ $\forall j,t,r,k$ (18)

$C_{max,k} \geq 0$ $\forall k$ (19)

As in SP_1 the objective function (11) represents minimizing the expected makespan.

Constraint set (12) ensures that each job is assigned to a position. Constraint set (13) guarantees that there is exactly one job in each position. Constraint set (14) forces the same sequence for all scenarios. Constraint set (15) is added for determining the set of jobs before the breakdown. It guarantees that summation of the processing times of the jobs completed before the breakdown does not exceed breakdown time. Constraint set (16) is related to the proper use of the positions. It ensures that later positions are used after the breakdown. The constraint together with the Constraint set (13) ensures that earlier positions are used first before the breakdown. Constraint set (17) defines the makespan values. Constraint sets (18) and (19) are binary and nonnegativity constraints, respectively.

3.4 An Example

We illustrate the model solution through 10-job and 3-scenario instance whose data are tabulated below.

Table 3. 1 Processing times of jobs in example

i	1	2	3	4	5	6	7	8	9	10
p_i	5	12	19	26	41	49	53	72	78	95

Table 3. 2 Breakdowns times and probabilities of an example

k	1	2	3
s_k	100	200	300
π_k	0.2	0.3	0.5

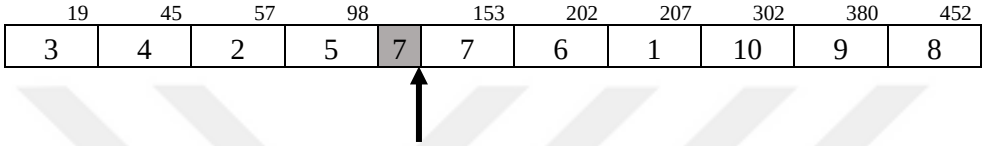
The breakdown durations (i.e. $d_k$values) are assumed to be zero.

Using the CPLEX solver, the GAMS software gives the following optimal job sequence:

3-4-2-5-7-6-1-10-9-8

- If scenario 1 occurs, the schedule will be implemented as:

Table 3. 3 Proposed schedule that shows the breakdown of scenario 1

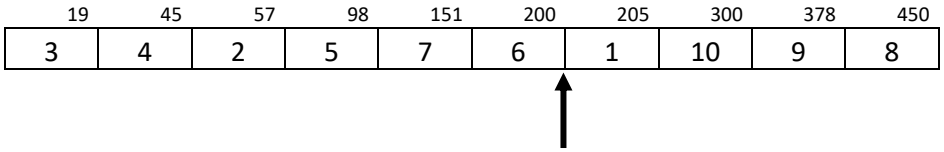


As the jobs are nonresumable, 2 units of job 7 before the breakdown are wasted (shown by a shaded box) and job 7 restarts after the breakdown and processed for 53 units. The resulting C_{max} value, i.e, $C_{max,1}$ is calculated as:

$$C_{max,1} = 100 + p_7 + p_6 + p_1 + p_{10} + p_9 + p_8 = 452$$

- If scenario 2 occurs, the schedule will be implemented as:

Table 3. 4 Proposed schedule that shows the breakdown of scenario 2

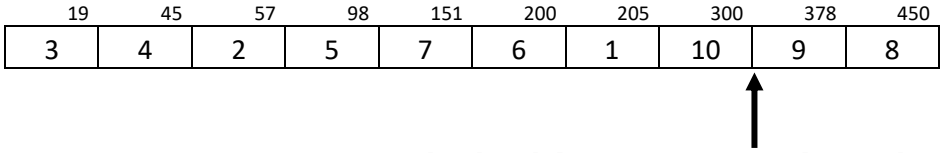


s_2 coincides with the completion of time Job 6, hence there will no idle time. $C_{max,2}$ value is then the sum of the processing times of all jobs, i.e.,

$$C_{max,2} = p_3 + p_4 + p_2 + p_5 + p_7 + p_6 + p_1 + p_{10} + p_9 + p_8 = 450$$

- If scenario 3 occurs, the schedule will be implemented as:

Table 3. 5 Proposed schedule that shows the breakdown of scenario 3



As in scenario 2, in scenario 3, the breakdown time coincides with a job completion, hence $C_{max,3}$ is the sum of the processing times of all jobs, i.e., 450.

Using the $C_{max,k}$ and π_k values, the optimal objective function values, z^*, is found as

$$z^* = \sum_k \pi_k * C_{max,k}$$

$$= \pi_1 * C_{max,1} + \pi_2 * C_{max,2} + \pi_3 * C_{max,3}$$

$$= 0.2 * 452 + 0.3 * 450 + 0.5 * 450$$

$$= 450.4$$

3.5 Expected Value Problem and the Value of the Stochastic Solution

The Expected Value Problem (EVP) is the deterministic counterpart of a Stochastic Problem (SP) where all the random parameters are replaced by their expected values. In other words, EVP can be considered an SP with a single scenario where the values of random parameters are set to their expected values. We refer to this single scenario as the expected value scenario (EV scenario).

When the problem is deterministic, i.e., when we have a single scenario as in the EVP, once the set of jobs to be processed before the breakdown is determined, any job sequence where this set is processed before the remaining ones would achieve the same objective function value for the EVP. Therefore, we can further simplify the EVP for the two-stage SP presented in Section 3.2 by removing the sequencing decisions, which would yield a knapsack problem where the capacity of the

knapsack is the expected breakdown time and the capacity usage values for the items (i.e., jobs) are the processing times. In compliance with this scheme, letting $\bar{s} = \sum_k \pi_k * s_k$ denote the expected breakdown and x_i be a binary variable which is 1 if job i is processed before the breakdown (0 otherwise), the corresponding EVP can be formulated as:

$$\max \sum_i p_i * x_i$$

Subject to

$\sum_i p_i * x_i \leq \bar{s}$

$x_i \in \{0,1\} \qquad \forall i$

By solving the above model, the optimal set of jobs to be processed (i.e., the jobs with $x_i^* = 1$) can be identified. A sequence where the jobs with $x_i^* = 1$ are processed before the jobs with $x_i^* = 0$ yields the same objective function value for the EVP. Without loss of generality, we use Shortest Processing Time (SPT) sequence within each group of jobs. The optimal objective function value of the EVP can be simply computed as the makespan for the EV scenario. The actual value of using a given sequence, which is known as the expected value of using the EV solution (EEV), can also be computed easily by considering the makespan for every scenario as:

$$z_{EEV} = \sum_k \pi_k * C_{max,k}$$

where π_k is the realization probability of scenario k and $C_{max,k}$ is the makespan under scenario k.

The Value of the Stochastic Solution (VSS) measures how well the SP solution performs with respect to the EVP solution and it is computed as the difference between the expected objective values attained by these solutions as:

$$VSS = z_{EEV} - z_{SP}^{*},$$

where z_{SP}^{*} is the optimal objective function of the SP. VSS shows the expected improvement in the objective function as a result of using a stochastic model rather than a deterministic model.

3.6 An Example

By using the data given in the example presented in Section 3.4, the expected breakdown is obtained as $\bar{s} = \sum_k \pi_k * s_k = 0.2 * 100 + 0.3 * 200 + 0.5 * 300 = 230$. The EVP based on this expected value scenario is solved and the following results are obtained:

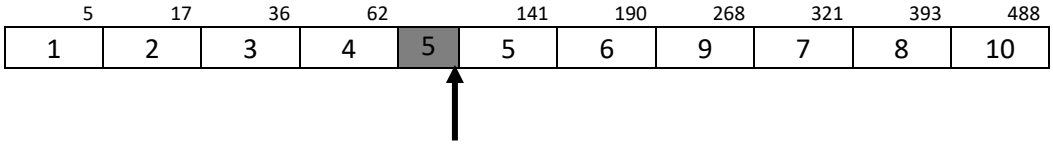
Table 3. 6 Output of the knapsack problem

Variables	$x_i^* = 1$	$x_i^* = 0$
Jobs	1	7
	2	8
	3	10
	4	
	5	
	6	
	9	

Using the solution of the EVP and considering the SPT sequence within each group, we obtain the sequence to be evaluated as 1-2-3-4-5-6-9-7-8-10. A detailed illustration of the realization of job completion times for each scenario is presented below.

EV solution for scenario 1:

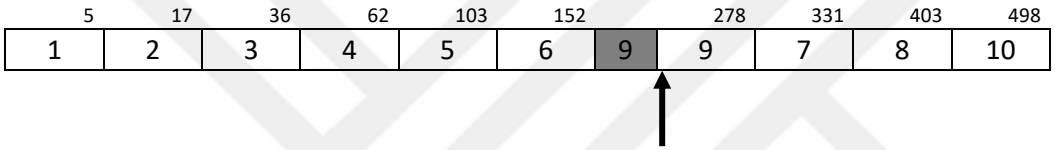
Table 3. 7 Schedule of EV solution for scenario 1



At time t=100, job 5 is interrupted and hence it is restarted after the breakdown. Makespan for this scenario is $C_{max,1} = 488$.

EV solution for scenario 2:

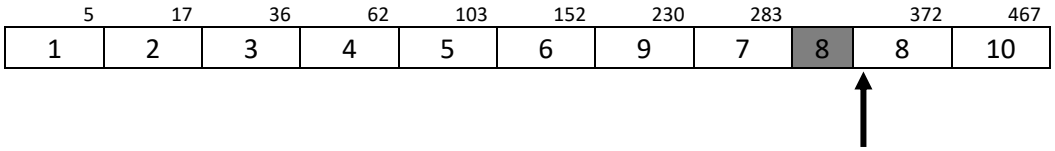
Table 3. 8 Schedule of EV solution for scenario 2



At time t=200, job 9 is interrupted and hence it is restarted after the breakdown. Makespan for this scenario is $C_{max,2} = 498$.

EV solution for scenario 3:

Table 3. 9 Schedule of EV solution for scenario 3



At time t=300, job 8 is interrupted and hence it is restarted after the breakdown. Makespan for this scenario is $C_{max,3} = 467$.

Then, EEV is calculated as follows:

$$z_{EEV} = \sum_{k} \pi_k * C_{max,k} = 0.2 * 488 + 0.3 * 498 + 0.5 * 467 = 480.5$$

From Section 3.4, the optimal objective function value of the SP solution is $z_{SP}^{*} = 450.4$. Then, the VSS can be computed as:

$$VSS = z_{EEV} - z^*_{SP} = 480.5 - 450.4 = 30.1$$

The expected objective value is reduced to 450.4, which corresponds to an improvement of 30.1 units by including stochasticity in the model. Although solving the SP rather than the EVP is computationally more challenging, the associated benefits could be significant as also illustrated by this example.

CHAPTER 4

BRANCH & BOUND METHOD

The aim of a branch and bound (B&B) algorithm is to find an optimal solution by reducing the search space of all feasible solutions using some mechanisms like dominance rules, lower and upper bounds.

In this chapter, we first give the basics of a B&B algorithm (Section 4.1). In section 4.2, we describe our branch and bound algorithm in detail.

4.1 Basics of Branch and Bound Method

The solution space is divided into feasible subspaces. The subspaces are called nodes, each of which is evaluated by estimates of the optimal objective function values, called bounds. The evaluation process completes when all nodes are implicitly or explicitly evaluated (visited or fathomed). At termination, the algorithm returns an optimal or a predefined satisfactory solution.

A typical sequencing problem requires the evaluation of n! complete solutions. A branch and bound algorithm evaluate those solutions implicitly as shown in the below figure.

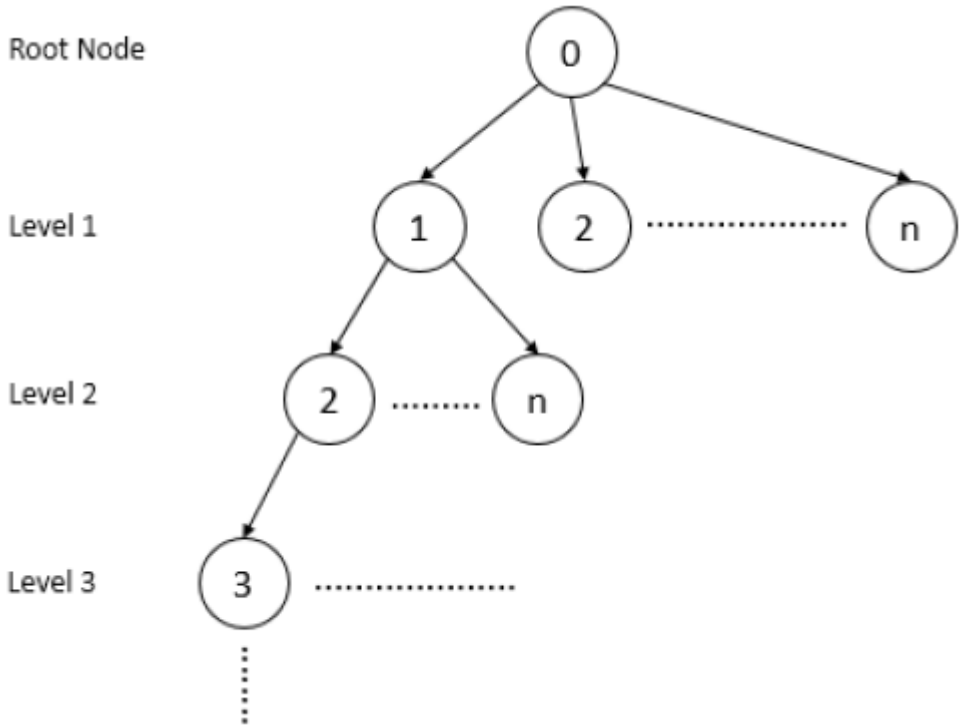


Figure 4.1 A partial branching tree

4.2 Proposed Branch and Bound Algorithm

All n! sequences of the single machine makespan problem are identical when there are no breakdowns. When there are breakdowns, many, but not all, of those sequences are identical. To avoid the generation of identical sequences, we propose an efficient branching scheme.

According to our branching scheme, we generate nodes representing the eligible jobs. We call a task, say job *j*, eligible for partial sequence (PS) if one of the following conditions holds:

i. $j > r$ when job r is the last sequenced job of PS.

ii. $\sum_{i \in PS} p_i + p_j > s_k$ where k satisfies $s_{k-1} < \sum_{i \in PS} p_i < s_k$

With this branching scheme, we avoid the repetition of many partial solutions. We clarify the branching scheme using a 5-job and 3-scenario example instance whose data are tabulated below.

Table 4.1 Processing time data

i	1	2	3	4	5
p_i	7	32	53	60	70

Table 4.2 Breakdown time data

k	1	2	3
s_k	47	100	140

Assume PS = {1}, jobs 2,3,4, and 5 can be added to PS as their indices are higher than that of job 1.

Now assume PS = {2}, jobs 3,4, and 5 can be added to PS as their indices are higher than that of job 1. Job 1 cannot be assigned as $1 \not\succ 2$ and $\sum_{i\in PS} p_i + p_1 = 32 + 7 < 47 = s_1$

Partial sequence {2,1} is not created as it is identical to {1,2}.

Now assume a partial sequence PS = {1,4} and $\sum_{i\in PS} p_i = p_1 + p_4 = 67$.

- Job 5 can be added as 5 > 4.
- Job 2 cannot be added as $2 \not\prec 4$ and $\sum_{i\in PS} p_i + p_2 = 67 + 32 = 99 < 100 = s_2$ where $s_1 = 47 < \sum_{i\in PS} p_i = 67 < 100$.

{1,4,2} is not created as it is identical to {1,2,4}.

- Job 3 can be added as:

$\sum_{i\in PS} p_i + p_3 = 67 + 53 = 120 > 100 = s_2$ where where $s_1 = 47 < \sum_{i\in PS} p_i < 100$.

{1,4,3} is created as it is not identical to {1,3,4}.

4.3 Bounding Mechanisms

Lower bounds are underestimates of the optimal objective function values. The efficiency of a branch and bound algorithm highly depends on the power of the lower bounds that help to fathom the nonpromising nodes without evaluating them. A partial schedule is eliminated value whenever its lower bound value is no less than the best known upper bound value. The upper bounds are overestimates of the optimal objective function values and they are usually found by evaluating any feasible solution to a problem.

In our study, we evaluate the following three feasible sequences:

i. Shortest Processing Time (SPT)
ii. Longest Processing Time (LPT)

We evaluate the above sequences and use the one that yields the minimum objective function value as an initial upper bound. The initial upper bound is updated whenever we find a complete solution with a smaller makespan value.

We develop two lower bounds using the idea of evaluating the sequences for each scenario. For a partial sequence PS, we found a lower bound on PS, $LB\ (PS)$ as follows:

$$LB(PS) = \sum_{k=1}^{m} \pi_k * LB\ (PS \mid k)$$

where $LB\ (PS \mid k)$ is a lower bound the makespan value given scenario k, i.e., a lower bound on a deterministic problem with a single scenario that occurs at time s_k. We take breakdown time, d_k, zero, as it is irrelevant to optimization.

To find $LB(PS)$, we use two approaches. The first approach, $LB_1(PS)$, is used as a filtering mechanism.

To find $LB_1(PS)$, through $LB_1(PS \mid k)$ values, we consider two cases.

Case 1. $s_k \leq \sum_{i \in PS} p_i$ then evaluate the PS by appending the unsequenced jobs in any order, let the resulting C_{max} value be $LB_1(PS \mid k)$.

PS is the given order, once the order is used s_k will be exceeded and the sequence of the unsequenced jobs becomes immaterial, altogether contributing to the lower bound as $\sum_i p_i$.

Case 2. $s_k > \sum_{i \in PS} p_i$ then we set $LB_1(PS \mid k) = \sum_i p_i$.

Hence we assume the jobs can be preempted as all sequences are immaterial each leading to a makespan value of $\sum_{i=1}^{n} p_i$.

We illustrate $LB_1(PS)$ via the following example instance with 5 jobs and 4 scenarios.

Tablo 4.3 Processing time of jobs in example

i	1	2	3	4	5
p_i	10	30	40	50	60

Tablo 4.4 Breakdown time and occurrence probability data

k	1	2	3	4
s_k	20	30	100	190
π_k	0.2	0.3	0.4	0.1

Assume PS={1,2}, $\sum_{i \in PS} p_i = 40$ and $\sum_{i \notin PS} p_i = 150$

- For k=1, $s_1 = 20 < \sum_{i \in PS} p_i = 40$

Hence case 1 occurs.

$$LB_1(PS \mid 1) = s_1 + p_2 + \sum_{i \notin PS} p_i = 20 + 30 + 150 = 200$$

- For k=2, $s_1 = 30 < \sum_{i \in PS} p_i = 40$

  Case 1 occurs.

$$LB_1(PS \mid 2) = s_2 + p_2 + \sum_{i \notin PS} p_i = 30 + 30 + 150 = 210$$

- For k=3, $s_3 = 100 > \sum_{i \in PS} p_i = 40$

  Case 2 occurs.

$$LB_1(PS \mid 3) = \sum_{i \in PS} p_i = 190$$

- For k=4, $s_4 = 190 > 40$

  Case 2 occurs.

$$LB_1(PS \mid 4) = \sum_{i \in PS} p_i = 190$$

The overall lower bound, $LB_1(PS)$ is found as:

$$LB_1(PS) = \sum_{k=1}^{4} \pi_k * LB_1(PS \mid k)$$

$$= 0{,}2 * 200 + 0{,}3 * 210 + 0{,}4 * 190 + 0{,}1 * 190$$

$$= 198$$

The second approach, $LB_2(PS)$, proceeds like $LB_1(PS)$ when $s_k \leq \sum_{i \in S} p_i$, however, improves $LB_1(PS)$ for case 2. In doing so, we let R_k be an upper bound on the number of jobs that can be processed before s_k, and we let I_k denote the time left for unsequenced jobs after the last job in PS is processed, accordingly $I_k = s_k - \sum_{i \in PS} p_i$ which is positive for case 2.

To find R_k, we use the following relation:

$\sum_{i=1}^{R_k} p_i \leq s_k$ and $\sum_{i=1}^{R_k+1} p_i > s_k$ where p_i is the i^{th} smallest processing in the unsequenced jobs set, i.e., $\overline{PS}$.

We consider four cases for R_k values.

Case 1. $R_k = 0$

$$LB_2(PS \mid k) = s_k + \sum_{i \notin PS} p_i$$

Case 2. $R_k = 1$

We find the largest p_i in $\overline{PS}$, that is no more than I_k, i.e.,

$$r = Max\{i \epsilon \overline{PS} \mid p_i \leq I_k\}$$

$$LB_2(PS \mid k) = s_k + \sum_{i \notin PS} p_i - p_r$$

Thus, we maximize the processing time put in I_k, thereby minimizing the makespan.

Case 3. $R_k = 2$

As in Case 2, we aim to maximize the processing put in I_k. This can be done in two ways with a single job or with a job pair.

A single job that maximizes the total processing should satisfy the following condition:

$$p_{[r]} = Max\left\{ p_{[i]} \,\middle|\, [i] \epsilon \overline{PS},\ p_{[i]} \leq I_k \leq p_{[i]} + p_{[1]} \right\}$$

Note that such [r] may not exist.

This condition follows that jobs [r+1], [r+2], … cannot fit in I_k and job[r] cannot fit in I_k together with another unsequenced job.

A job pair that maximizes the total processing can be found through the following procedure.

Step 0. If [r] does not exist, [r] is the index of the longest job that fits in I_k in $\overline{PS} + 1$.

$$PSS = \overline{PS}$$

$$P = p_{[r]}$$

Step 1. Find the longest job in PSS that fits in I_k together with job[r], hence find job[a] that satisfies the following condition:

$$p_{[a]} = Max\left\{p_{[i]} \middle| [i] \in PSS, p_{[i]} + p_{[r-1]} \leq I_k\right\}$$

If such a job [a] does not exist, go to *Step 3.*

$$p = \max\{p,\ p_{[a]} + \ p_{[r-1]}\}$$

Step 2. PSS = PSS \{[1],…[a]}

The unsequenced jobs set is updated as the shorter jobs cannot fill I_k better.

$$r = r - 1$$

Go to *Step 1*.

Step 3. Stop

$$LB_2(PS \mid k) = s_k + \sum_{i \in \overline{PS}} p_i - p$$

Case 4. $k^* \geq 3$

Case 4.1. The longest unsequenced R_k tasks can fit to I_k

$\sum_{i=1}^{R_k} p'_{[i]} < I_k$ where $p'_{[i]}$ is the longest unscheduled job that can fit in I_k,i.e., $p'_{[i]} < I_k$ for all i.

$$LB_2(PS \mid k) = s_k + \sum_{i \in PS} p_i - \sum_{i=1}^{R_k} p'_{[i]}$$

Case 4.2. $\sum_{i=1}^{R_k} p'_{[i]} \geq I_k$

$$LB_2(PS \mid k) = \sum_{i=1}^{n} p_i$$

In our experiments, we first find $LB_1(PS)$. If $LB_1(PS) \geq UB$ then we fathom PS. Otherwise, we find $LB_2(PS)$ and fathom PS if $LB_2(PS) \geq UB$.

Batun ve Azizoğlu (2009) used the k^* values in their deterministic single machine total flow problem with several breakdowns. They use the procedure of our Case 4 for all k^* values. Thus, we improve the lower bound of Batun ve Azizoğlu (2009) by providing more efficient bounds for the special cases.

We illustrate $LB_2(PS)$ through a 7-job, 6-scenario example instance whose data are tabulated below.

Table 4.5 Processing time data for 7-job, 6-scenario example instance

i	1	2	3	4	5	6	7
p_i	10	15	20	25	60	75	90

Table 4.6 Breakdown time and occurrence probability data for 7-job, 6-scenario example instance

k	1	2	3	4	5	6
s_k	20	40	65	120	210	295
π_k	0.1	0.2	0.3	0.2	0.1	0.1

Assume PS={1,2}:

$\sum_{i\in PS} p_i = 10 + 15 = 25$ and $\sum_{i\notin PS} p_i = 20 + 25 + \cdots + 90 = 270$

- For k=1

$$s_1 = 20 < \sum_{i\in PS} p_i = 25$$

$$LB_2(PS \mid 1) = 20 + 15 + 270 = 305$$

- For k=2

$$s_2 = 40 > \sum_{i\in PS} p_i = 25$$

$$I_2 = s_2 - \sum_{i\in PS} p_i = 40 - 25 = 15$$

$p_3 = 20 > 15$ ➔ $\boldsymbol{R_2 = 0}$

$$LB_2(PS \mid 2) = 40 + 270 = 310$$

As $s_2 > \sum_{i\in PS} p_i$, $s_k > \sum_{i\in PS} p_i$ because $s_k's$ are in their increasing order, so we do not check, $s_k > \sum_{i\in PS} p_i$ for later scenarios.

- For k=3

$$I_3 = s_3 - \sum_{i\in PS} p_i = 65 - 25 = 40$$

$p_3 = 20 < 40$ $\quad$ $p_3 + p_4 = 20 + 25 > 40$ ➔ $R_3 = 1$

$$Max\{i\epsilon\overline{PS} \mid p_i \leq I_2\} = Max\{p_3, p_4\} = 25$$

$$LB_2(PS \mid 3) = 65 + 270 - 25 = 310$$

- For k=4

$$I_4 = s_4 - \sum_{i\in PS} p_i = 120 - 25 = 95$$

$p_3 + p_4 = 20 + 25 < 95$

$p_3 + p_4 + p_5 = 105 > 95$ ➔ $\boldsymbol{R_4 = 2}$

We apply the procedure find the jobs that could only fit in I_4 alone.

$p_7 = 90 < 95$ $p_7 + p_3 = 90 + 20 = 110 > 95$

Job 7 fits alone, P=90.

Take p_6. $p_6 + p_3 = 95$ $p_6 + p_4 = 100 > 95$

$p = \max\{90,95\} = 95$

Take p_5. $p_5 + p_3 = 80 < 95$ $p_5 + p_4 = 85 < 95$

$p = \max\{95,85\} = 95$

$LB_2(PS \mid 4) = 120 + 270 - 95 = 295$

- <u>For k=5</u>

$$I_5 = s_5 - \sum_{i \in PS} p_i = 210 - 25 = 185$$

$p_3 + p_4 + p_5 + p_6 = 180 < 185$

$p_3 + p_4 + p_5 + p_6 + p_7 = 270 > 185$ ➔ $\boldsymbol{R_5 = 4}$

$$\sum_{i=1}^{4} p'_{[i]} = p_7 + p_6 + p_5 + p_4 = 250 > I_5$$

$$LB_2(PS \mid 5) = \sum_{i=1}^{n} p_i = 295$$

- <u>For k=6</u>

$$I_6 = s_6 - \sum_{i \in PS} p_i = 295 - 25 = 270$$

$\sum_{i \notin \overline{PS}} p_i = 270$ ➔ All jobs in $\overline{PS}$ fits.

$LB_2(PS \mid 6) = 295$

$LB_2(PS) = \sum_{k=1}^{6} \pi_k * LB_2(PS| k) = 0.1 * 305 + 0.2 * 310 + 0.3 * 310 + 0.2 * 295 + 0.1 * 295 + 0.1 * 295 = 303.5$

CHAPTER 5

COMPUTATIONAL EXPERIMENTS

In this chapter, we discuss the results of our experiment that is designed to test the performances of the stochastic programming models and the branch and bound algorithm. We also assess the value of using a stochastic programming approach, i.e., the value of the stochastic solution.

In Section 5.1, we report our data generation scheme. Section 5.2 defines our performance measures. In section 5.3, we discuss the computational results through our performance measures.

5.1 Data Generation

- We set the number of jobs.
  - n = 10 and 15 for the stochastic programming model
  - n = from 10 to 50 for the branch and bound algorithm
- We have two sets for processing times, p_i values. In the first set, Set 1, processing times are generated from a discrete uniform distribution between 1 and 10, i.e., U [1,10]. In the second set, Set 2, U [1,100] is used to generate processing times. Note that Set 2 resides higher processing times in a wider range compared to Set 1.

We assume that the durations of breakdown values are zero as those times are irrelevant for makespan optimization.

- We set the number of scenarios, *m*, to 3 and 5. The breakdown times, s_k values, are generated according to two plans:

Plan 1. Periodic s_k values

In this plan, s_k is set to $\frac{\sum_i p_i}{m} * k$, i.e, $s_1 = \frac{\sum_i p_i}{m}$, $s_2 = 2 * \frac{\sum_i p_i}{m}$, …, $s_m = \sum_i p_i$.

- Note that s_m corresponds to the end of all processing, i.e., no breakdown case

Plan 2. Random s_k values for the first m-1 scenarios.

s_k values are generated randomly between $max\{p_i\}$ and $\sum_i p_i - 1$. We use $max\{p_i\}$ as the earliest time to guarantee that all jobs can be processed before the breakdown.

For the last scenario, we take $s_m = \sum_i p_i$ to mean that no breakdown has occurred.

- We use two sets for the scenario probability, i.e., π_k values, as:

Set 1: All scenarios are equally likely to occur, i.e, $\pi_k = \frac{1}{m}$ for all k.

Set 2: The later breakdowns are more likely.

For 3-scenario case, we set: $\pi_1 = \frac{1}{6}$, $\pi_2 = \frac{2}{6}$, $\pi_3 = \frac{3}{6}$

For 5-scenario case, we set: $\pi_1 = \frac{1}{15}$, $\pi_2 = \frac{2}{15}$, $\pi_3 = \frac{3}{15}$, $\pi_3 = \frac{4}{15}$, $\pi_5 = \frac{5}{15}$

For each n, we have two processing time sets, two breakdown plans, two probability sets, and two values of *m*, hence 2x2x2x2=16 combinations.

We plan 2 different values of n for the model and 5 different values for the algorithm of n, yielding a total of 16x2=32 and 16x5=80 combinations for each solution method, respectively. For each combination, we plan to generate 10 instances. Hence our experiment set will have 320 problem instances for the model and 800 problem instances for the algorithm.

5.2 Performance Measures

We evaluate the performance of the stochastic models by the CPU (Central Processing Units) seconds. We report average a maximum, i.e., worst case, CPU times.

For the B&B, we will use the CPU times and the number of nodes as the performance measures. Average and maximum CPU times and number of nodes will be reported.

To assess the value of the stochastic programming approach, we evaluate the expected makespan value of the expected value solution (EEV) and the value of the stochastic solution (VSS). We report average and maximum EEV-SP values for each problem combination.

We set a termination limit of two hours both for the mathematical models and branch and bound algorithm. We also report the number of instances that could be solved to optimality in two hours.

The mathematical models are solved by CPLEX Optimizer 20.1.0 We code the branch and bound algorithm using the C++ programming language. All experiments are conducted on a personal computer with 11th Gen Intel® Core™ i5 -1135G7 @2.40GHz (4CPUs), 2.42GHz processors, and 8GB RAM.

5.3 Analysis of the Results

In this section, we report on the performances of our mathematical models and branch and bound algorithm (B&B). Section 5.3.1 discusses the performance of the stochastic models, Precedence Based Stochastic Model (SP_1) and the Position Based Stochastic Model (SP_2). Results of the B&B are revealed in the Section 5.3.2, and discussion about these results are made in this section. In addition, VSS results are discussed in the Section 5.3.3.

5.3.1 Stochastic Programming

In this subsection, we report on the relative performances of SP_1 and SP_2 for n=10. Tables 5.1 and 5.2 compare SP_1 and SP_2 for low processing time ($p_i \sim U[1,10]$) and high processing time ($p_i \sim U[1,100]$), respectively. The tables report on the average and maximum CPU times, for m=3, 5 and two probability sets.

Table 5. 1 Performances of *SP1* and *SP2*, $p_i \sim U[1,10]$, n=10

m	**Probability (1=Equal, 2=Later)**	**SP1**		**SP2**	
		CPU Time		**CPU Time**	
		Average	**Maximum**	**Average**	**Maximum**
3	1	11.2	30	1.1	3
	2	8.2	12	18.8	187

Table 5. 2 Performances of *SP1* and *SP2*, $p_i \sim U[1,100]$, n=10

m	**Probability (1=Equal, 2=Later)**	**SP1**		**SP2**	
		CPU Time		**CPU Time**	
		Average	**Maximum**	**Average**	**Maximum**
3	1	5.7	14	612.7	1964
	2	5.1	8	292.8	1745
5	1	126.7	969	2002.2	3514
	2	89.4	217	2451.9	6101

As can be observed from Tables 5.1 and 5.2, the performances of both models deteriorate as m increases. This is due to the increase of the binary variables with increase in m, for both SP_1 and SP_2. We also observe that the performance of SP_1 is better than that of SP_2 which can be attributed to the fewer binary variables used by the former model, SP_1.

Attributing to the its superior performance particularly for $p_i \sim U[1,100]$, we continue the parametric analysis with SP_1 in the next subsection.

We now report on the performance of the precedence based stochastic model, SP_1, on larger-sized problems. Table 5.3 gives the performance for $p_i \sim U[1,100]$. The table give the average and maximum CPU times and the number of unsolved instances in two hours. We try on different values of n starting with *n*=10, in increments of 5.

Table 5. 3 CPU Times of SP1, $p_i \sim U[1,100]$

n	**m**	**Probability (1=Equal, 2=Later)**	**CPU Time**		**Number Solved**
			Average	**Maximum**	
10	3	1	8.8	30	10
		2	6.4	18	10
	5	1	112.2	641	10
		2	29.1	80	10
15	3	1	3617	7200	1
		2	-	-	-
	5	1	-	-	-
		2	-	-	-

Note from the above tables that when *n*=10, all instances can be solved in 2 hours. The average CPU time is about 9 seconds when *m*=3 and about 110 minutes when *m*=5. The maximum CPU time is 10 minutes and observed in equal probability combination. When n becomes 15, the majority of the instances are left unsolved in 2 hours and the associated entries in Table 5.3 are left empty. The increase in the complexity of the solutions with increase in *n* is due to the increase in the number of binary variables in SP_1. As *n* increases, the number of binary variables associated to *n* increases exponentially.

5.3.2 Branch & Bound Algorithm

In this section, we report the results of the proposed branch and bound algorithm. We consider the average and maximum CPU times (in seconds), the average and the maximum number of nodes evaluated, and the number of unsolved instances within the two-hour time limit as our performance measures. We report these measures for five levels of the number of jobs (n = 10, 20, 30, 40, and 50) and two levels of the number of scenarios (m = 3 and 5) in Tables 5.4-5.11. The number of unsolved instances, out of 10 instances within the time limit are presented together with maximum CPU times and denoted in parentheses.

Table 5. 4 Computational Results for the Branch and Bound Algorithm for Set 1, Plan 1 and $p_i \sim U[1,10]$

n	m=3				m=5			
	CPU Time		Number of Nodes		CPU Time		Number of Nodes	
	Average	Maximum	Average	Maximum	Average	Maximum	Average	Maximum
10	0.011	0.014	74	117	0.036	0.087	983	3,208
20	0.033	0.042	210	211	0.605	3038	7,922	39,860
30	0.115	0.141	464	466	0.879	7463	10,931	104,848
40	0.177	0.215	817	821	0.195	0.264	830	889
50	0.249	0.269	1,275	1,276	0.270	0.359	1,298	1,523

Table 5. 5 Computational Results for the Branch and Bound Algorithm for Set 1, Plan 2 and $p_i \sim U[1,10]$

n	m=3				m=5			
	CPU Time		Number of Nodes		CPU Time		Number of Nodes	
	Average	Maximum	Average	Maximum	Average	Maximum	Average	Maximum
10	0.026	0.045	264	1,216	0.056	0.139	950	3,843
20	722.036	7200 (1)*	5,088,702	49,819,874	2.666	24.904	73,820	728,869
30	0.211	0.231	447	466	737.423	7200 (1)*	2,991,954	27,717,338
40	4.126	38.359	62,426	615,837	2598.080	7200 (3)*	18,616,526	88,196,192
50	0.436	0.485	1,289	1,444	1440.231	7200 (2)*	9,732,598	60,060,234

*The figures in the parentheses give the number of unsolved instances in 2 hours.

Table 5.6 Computational Results for the Branch and Bound Algorithm for Set 2, Plan 1 and $p_i \sim U[1,10]$

n	m=3				m=5			
	CPU Time		Number of Nodes		CPU Time		Number of Nodes	
	Average	Maximum	Average	Maximum	Average	Maximum	Average	Maximum
10	0.087	0.104	85	135	0.296	0.637	2,913	11,424
20	0.220	0.404	569	3,668	0.534	3.072	5,806	48,618
30	0.267	0.336	465	466	11.844	100.610	268,391	2,422,754
40	0.373	0.620	814	821	0.217	0.301	865	1,044
50	0.506	0.590	1,271	1,276	69.763	515.374	581,422	4,245,445

Table 5. 7 Computational Results for the Branch and Bound Algorithm for Set 2, Plan 2 and $p_i \sim U[1,10]$

n	m=3				m=5			
	CPU Time		Number of Nodes		CPU Time		Number of Nodes	
	Average	Maximum	Average	Maximum	Average	Maximum	Average	Maximum
10	0.122	0.220	370	3,023	0.646	2.605	19,315	108,439
20	0.407	2.483	7,686	72,036	1443.159	7200 (2)*	40,819,739	206,208,655
30	0.323	0.843	1,338	8,032	8.060	76.215	186,921	1,851,968
40	0.327	0.408	853	1,143	721.237	7200 (1)*	11,532,788	115,233,391
50	0.463	0.576	1,267	1,276	12.32	118.429	209,370	2,080,665

*The figures in the parentheses give the number of unsolved instances in 2 hours.

Table 5. 8 Computational Results for the Branch and Bound Algorithm for Set 1, Plan 1 and $p_i \sim U[1,100]$

n	m=3				m=5			
	CPU Time		Number of Nodes		CPU Time		Number of Nodes	
	Average	Maximum	Average	Maximum	Average	Maximum	Average	Maximum
10	0.053	0.102	500	1,526	0.130	0.201	1,366	2,830
20	0.048	0.072	339	676	29.001	216.975	648,147	5,040,425
30	0.079	0.116	471	523	0.470	1.490	7,302	25,474
40	0.108	0.129	833	871	723.572	7200 (1)*	5,778,795	57,245,222
50	0.165	0.181	1,306	1,370	0.285	0.540	1,970	3,699

*The figures in the parentheses give the number of unsolved instances in 2 hours.

Table 5. 9 Computational Results for the Branch and Bound Algorithm for Set 1, Plan 2 and $p_i \sim U[1,100]$

n	m=3				m=5			
	CPU Time		Number of Nodes		CPU Time		Number of Nodes	
	Average	Maximum	Average	Maximum	Average	Maximum	Average	Maximum
10	0.059	0.231	1,639	8,325	0.105	0.205	2,164	6,422
20	0.068	0.154	612	2,806	2347.972	7200 (3)*	47,376,972	180,590,045
30	722.469	7200 (1)*	10,396,335	103,226,591	1457.592	7200 (2)*	15,308,087	109,886,724
40	1440.100	7200 (2)*	12,367,343	65,342,289	2162.353	7200 (3)*	20,688,532	73,484,684
50	1440.190	7200 (2)*	15,966,518	84,327,441	2884.418	7200 (4)*	32,662,854	121,405,021

*The figures in the parentheses give the number of unsolved instances in 2 hours.

Table 5. 10 Computational Results for the Branch and Bound Algorithm for Set 2, Plan 1 and $p_i \sim U[1,100]$

n	m=3				m=5			
	CPU Time		Number of Nodes		CPU Time		Number of Nodes	
	Average	Maximum	Average	Maximum	Average	Maximum	Average	Maximum
10	0.132	0.174	825	1,730	0.304	0.395	2,530	6,154
20	0.264	1.334	2,645	22,369	834.931	4876.442	10,138,230	54,407,771
30	0.182	0.263	485	539	3.377	22.388	59,304	444,366
40	0.248	0.295	856	982	0.330	1334.000	1,714	8,544
50	0.304	0.364	1,290	1,396	0.325	0.484	1,431	2,174

Table 5. 11 Computational Results for the Branch and Bound Algorithm for Set 2, Plan 2 and $p_i \sim U[1,100]$

n	m=3				m=5			
	CPU Time		Number of Nodes		CPU Time		Number of Nodes	
	Average	Maximum	Average	Maximum	Average	Maximum	Average	Maximum
10	0.113	0.214	497	3,366	0.328	0.720	3,507	12,840
20	0.149	0.199	288	559	730.500	7200 (1)*	14,149,241	138,751,079
30	720.224	7200 (1)*	8,396,481	83,955,421	2161.599	7200 (3)*	25,664,053	95,446,921
40	860.591	7200 (1)*	10,279,600	71,293,331	2880.523	7200 (4)*	23,758,292	101,234,212
50	2.043	16.534	13,333	119,461	1569.596	7200 (2)*	10,637,254	96,322,856

*The figures in the parentheses give the number of unsolved instances in 2 hours.

5.3.2.1 Effects of the Problem Size Parameters

The tables in the Part 5.3.2 altogether reveal that the performance of the B&B deteriorates with increases in the number of jobs (*n*) and increases in the number of scenarios (*m*). The effect of *m* on the performance is more significant than that of n. For example, as can be observed from Table 5.6, the average CPU times, increases from 0.296 seconds to 69.763 seconds when *n* increases from 10 to 50 when *m*=5 for Set 2 instances with Plan 1 and $p_i \sim U[1,10]$. However, consistency in CPU times is not observed in some settings due to few instances. For example, as can be observed from Table 5.5, the average CPU times decreases from 722.036 seconds to 0.436 seconds when n increases from 20 to 50 for Set 1 instances and *m*=3 with Plan 2 and $p_i \sim U[1,10]$. When two computationally challenging instances that can be accepted as outliers are excluded from calculations, the average CPU times of remaining 8 instances become 0.131 seconds, which supports the statement that an increase on the CPU time is observed when the number of jobs increases. Like CPU times, the number of nodes generated also increases with the number of jobs.

There is no unsolved instance in two hours when *n*=10. However, the unsolved instances are observed with the increase in the of number jobs, mostly when *n*=40 and 50. The increases in the CPU times and the number of nodes is due to the inflated size of the branch and bound tree. The tree has *n* levels of depth.

On the other hand, the increases are not too significant, and the exponential nature of the search is dispelled by powerful branching scheme and bounding mechanisms. To check the performance for many more jobs, we solve instances with 100, 150 and 200 jobs when there are 5 scenarios and $p_i \sim U[1,100]$. We report the results in Table 5.12 (for Set 1 and Plan 1), Table 5.13 (for Set 1 and Plan 2), Table 5.14 (for Set 2 and Plan 1) and Table 5.15 (for Set 2 and Plan 2). The tables give the average and maximum CPU times and average and maximum number of nodes over 5 generated instances.

Table 5. 12 Computational Results for the Branch and Bound Algorithm for Set 1, Plan 1, higher *n* values and $p_i \sim U[1,100]$.

n	m=5			
	CPU Time		Number of Nodes	
	Average	Maximum	Average	Maximum
100	0.983	1.206	5,085	5,565
150	3.935	4.159	11,255	11,326
200	6.249	7.116	20,070	20,101

Table 5.13 Computational Results for the Branch and Bound Algorithm for Set 1, Plan 2, higher *n* values and $p_i \sim U[1,100]$.

n	m=5			
	CPU Time		Number of Nodes	
	Average	Maximum	Average	Maximum
100	2.063	2.385	5,042	5,050
150	2.995	3.123	11,326	11,326
200	23.147	88.293	31,192	75,735

Table 5. 14 Computational Results for the Branch and Bound Algorithm for Set 2, Plan 1, higher *n* values and $p_i \sim U[1,100]$.

n	m=5			
	CPU Time		Number of Nodes	
	Average	Maximum	Average	Maximum
100	1.664	1.843	5,037	5,047
150	2.896	2.955	11,285	11,326
200	5.087	5.172	20,054	20,101

Table 5. 15 Computational Results for the Branch and Bound Algorithm for Set 2, Plan 2, higher n values and $p_i \sim U[1,100]$.

n	m=5			
	CPU Time		Number of Nodes	
	Average	Maximum	Average	Maximum
100	1441.338	7200 (1)*	5572	7271
150	6.225	14.173	12630.6	17951
200	1444.133	7200 (1)*	20073.8	20101

*The figures in the parentheses give the number of unsolved instances in 2 hours

As can be noted from the tables, there are 2 unsolved instances out of 60 instances, and the CPU times for small instances are mostly less than 20 seconds. We observe that the number of jobs, *n*, affects the results quietly. The reason behind this situation is that our algorithms find the best optimal solution that is generally equal to the sum of the processing times of jobs with the powerful LBs easily. When the best optimal solution is found, fathoming structure helps to eliminate worse solutions.

Tables 5.4 through 5.11 show that the performance of the algorithm is better when *m* is smaller. This holds over all problem sets. Note that, when *n*=20 and $p_i \sim U[1,100]$ for Set 1 and Plan 2 instances, the maximum CPU time is 0.154 seconds when *m*=3 and 7200 seconds when *m*=5. The same behaviour also observed for the average CPU times. For *m*=5 combination, there are 4 unsolved instances in two hours. The same behaviour also observed for the average CPU times. Another notable example is when *n*=20 and $p_i \sim U[1,10]$ for Set 2 and Plan 2 instances. For this combination, the maximum CPU time is 2.483 seconds when *m*=3 and there are 2 unsolved instances when *m*=5. Likewise, number of nodes explored till the optimal solution is reached increases when *m* value changes from 3 to 5. For instance, the average number of nodes significantly increases from 485 to

59,304 when the *m* changes from 3 to 5 for *n*=30 in Table 5.10. This is due to the superior performance of the lower bounds for small *m*. Our lower bounds find an underestimate for each scenario and sum them to get the overall values. This follows, their performance deteriorates as the number of underestimates in the sum, increases.

5.3.2.2 Effects of the Processing Time Distributions

There are two sets for processing times, generated from a discrete uniform distribution $U[1,10]$ and $U[1,100]$ respectively. Discussion of the results is given as a bulleted list for each combination, and comments about results are made.

- Set 1 - Plan 1: There is not a significant difference or pattern in CPU times and number of nodes generated between the results of $p_i \sim U[1,10]$ and $p_i \sim U[1,100]$ for the *m*=3. Otherwise, our algorithm performs better for the $p_i \sim U[1,10]$ when *m*=5. For a notable example, while average CPU times for *n*=20, $p_i \sim U[1,100]$, Set 1 and Plan 1 is equal to 29.001 seconds, it is only 0.605 seconds for $p_i \sim U[1,10]$, approximately 15 times. In addition, 1 from 50 instances are not solved in the termination limit, two hours for $p_i \sim U[1,100]$. The same behavior with CPU times is observed for the number of nodes generated in the model.
- Set 1 – Plan 2: For this problem settings, there are many unsolved instances for both scenarios' number. By considering number of unsolved instances, performance of the branch and bound algorithm deteriorates for $p_i \sim U[1,100]$ with respect to $p_i \sim U[1,10]$.
- Set 2 – Plan 1: Results reveal that algorithm performs well in terms of number of nodes and computation times for $p_i \sim U[1,100]$ when *n* is relatively small for *m*=3. When *n* becomes taking higher values, results of $p_i \sim U[1,10]$ is

better than results of $p_i \sim U[1,100]$. However, performance measures are very close to ignore the results of this problem setting.

- Set 2 – Plan 2: Effects of processing time generation is more obvious for Set 2 and Plan 2 than other problem settings. Branch and bound algorithm perform poorer for $p_i \sim U[1,100]$ in terms of both of performance measures.

The instances with $p_i \sim U[1,10]$ produce very close objective function values for different job sequences due to similarity of the processing times. This follows solution found at the early levels of branching are close to the optimal expected makespan values and the B&B does not have search many more nodes. When $p_i \sim U[1,100]$ the sequences are much apart from each other and optimal solution can be reached after visiting many sequences having not so close expected makespan values.

Another point that should be emphasized is that we observe that the B&B for $p_i \sim U[1,100]$ instances are more sensitive to the use of LB_2 In other words, LB_2 makes more significant eliminations when $p_i \sim U[1,100]$. For example, the average CPU time for m= 5, Set 2 and Plan 1 decreases from 4.513 seconds to 3.949 seconds for $p_i \sim U[1,10]$ while it reduces from 135.151 seconds to 11.668 seconds for $p_i \sim U[1,100]$ in the Tables 5.20 and 5.22. This is due to the fact that LB_2 finds fewer chances to fill the intervals thereby resulting in higher idle time values when $p_i \sim U[1,100]$. Higher idle time values mean higher lower bounds, hence higher chances for node eliminations.

5.3.2.3 Effects of the Breakdown Plans

Breakdown times are determined according to two plans, periodic breakdowns named Plan 1 and random breakdowns named Plan 2. The performance of the algorithm for CPU time is better when breakdown times are determined as periodic with respect to random breakdown generally. For a notable example, the average

CPU times increase from 0.177 seconds to 4.126 seconds, approximately 40 times, after changing the breakdown plan from periodic to random for Set 1 instance with n=40, m=3 and $p_i \sim U[1,10]$. However, we observe the opposite of this indication where Plan 2 performs better than Plan 1. From Table 5.6 and Table 5.7, the average CPU time of plan 2 is better than the average CPU time of plan 1 when m=3, set 2 and n=40 and 50. However, these specific cases are insignificant because the difference between the average CPU time of plans is not remarkable. When we look at combinations in which n is greater than 50, the performance of algorithms deteriorates as the plan changes from 1 to 2, excluding the only combination of n=150, m=5 and set 1. To clarify, the average CPU time increases from 6.249 seconds to 23.147 seconds when n=200, m=5 and Set 1. In addition, the maximum CPU time shows the same behaviour that can be observed for the average CPU time. The average CPU times of the algorithm are significantly better for Plan 1 combinations than Plan 2 combinations, excluding the combinations of n=20, Set 2 and $p_i \sim U[1,100]$. The reason is that lower bound 2 (LB_2) considers minimizing the idle time, which is the difference between the breakdown time and the start time of the job that is interrupted by the breakdown. It is better to have more jobs that can be positioned in this idle time. The increment in the breakdown time from one scenario to another is determined periodically in Plan 1 (as opposed to being randomly determined in Plan 2), and hence is large enough to fill efficiently, particularly when the number of scenarios is low. When the situation is considered from this perspective, it becomes reasonable that combinations of Plan 1 perform better than the combinations of Plan 2.

There are big differences between breakdown plans in terms of number of unsolved instances in two hours. Below is a table that shows the total number of unsolved instances for 100 instances.

Table 5. 16 Distribution of unsolved instances to specific problem settings

Settings	**Plan 1 & *m*=3**	**Plan 1 & *m*=5**	**Plan 2 & *m*=3**	**Plan 2 & *m*=5**
Set 1 - $p_i \sim$[1,10]	0	0	1	6
Set 1 - $p_i \sim$[1,100]	0	1	5	12
Set 2 - $p_i \sim$[1,10]	0	0	0	3
Set 2 - $p_i \sim$[1,100]	0	0	2	10

As the results in Table 5.16 reveal, our algorithm performs well when breakdown times are determined periodically for both *m*=3 and *m*=5. Particularly, the effect of breakdown plan is observed as dramatically significant when the number of scenario increases.

Another point that should be emphasized is related to the number of nodes generated in the algorithm. The dramatic increase in the average and maximum number of nodes in almost all combinations excluding the same cases that behave differently than general cases while discussing average CPU time is observed. For example, average number of nodes generated increase from 7,922 to 73,820 between plan 1 and plan 2 when $p_i \sim U[1,10]$ *n*=20, *m*=5 and Set 1. For the cases that involve unsolved instances, this difference becomes huge. Therefore, Plan 2 performs worse than Plan 1 with respect to the number of nodes.

It can be summarized that breakdown plans have significant effects on the results of our algorithm in terms of our performance measures and Plan 1 performs well with respect to Plan 2 in our algorithm.

5.3.2.4 Effects of the Scenario Probabilities

There are two sets for the probability of scenarios. Set 1 defines the case where all scenarios are equally likely to occur, and Set 2 refers to the case where the later

breakdowns are more likely to occur. Our results show that CPU times are slightly higher for Set 2 in various combinations. However, almost all increases are negligible. We have a similar observation for the number of nodes and for the number of unsolved instances.

In the implementation of our lower bounds, each scenario is considered separately. A lower bound on the makespan is obtained for each scenario, and the probability values only affect the expected objective value. Therefore, the guidance of the bounds and the fathoming structure remains unchanged and the scenario probabilities do not have notable impact on the performance of algorithm.

5.3.2.5 Effects of the Lower Bounds

In this section, we discuss the effects of lower bounds on the performance of the proposed branch and bound algorithm. To see the effects of the lower bounds on the performance of the branch and bound algorithm, we compare the algorithm that uses both lower bounds and report the results in Tables 5.17 and 5.18, that tabulates the CPU times and the number of nodes for n=10.

Table 5. 17 Results of Performance Measure when LBs are not used for *n*=10

Job	Scenario	Set	Plan	CPU Time		Number of Nodes	
				Average	Maximum	Average	Maximum
				Without LBs			
10	3	1	1	3.837	6.581	662,574	1,193,225
			2	2.811	4.727	437,691	727,092
		2	1	2.967	3.762	678,120	919,323
			2	2.705	4.667	612,289	1,184,510
10	5	1	1	5.276	7.908	862,217	1,342,271
			2	4.775	7.492	773,654	1,298,581
		2	1	6.462	12.262	1,454,319	2,926,585
			2	4.295	10.677	908,034	2,641,093

Table 5. 18 Results of Performance Measure when LBs are used for *n*=10

Job	Scenario	Set	Plan	CPU Time		Number of Nodes	
				Average	Maximum	Average	Maximum
				With LBs			
10	3	1	1	0.053	0.102	500	1,526
			2	0.059	0.231	1,639	8,325
		2	1	0.132	0.174	825	1,730
			2	0.113	0.214	497	3,366
10	5	1	1	0.130	0.201	1,366	2,830
			2	0.105	0.205	2,164	6,422
		2	1	0.304	0.395	2,530	6,154
			2	0.328	0.720	3,507	12,840

Table 5.17 and 5.18 reveals that the number of nodes generated, and the CPU time improve significantly when the lower bounds are incorporated. The effect of the lower bounds is more significant when there are more scenarios. For example, the average number of nodes for *m*=3, Set 2 and Plan 2 is 4,967 and 612,289 when the lower bounds are used and they are not used, respectively. In addition to *m*=3, using the lower bounds decreases the number of nodes from 908,034 to 3,507 for the same combination when *m*=5. This strong power of the lower bounds dispels the exponential nature of the search. The same inferences as the number of nodes generated can be made for the CPU times.

To show the effect of the stronger lower bound, i.e., LB_2 we design two branch and bound algorithms: one that uses LB_2 and does not use LB_2. Table 5.19 through Table 5.22 report the performance results. Tables 5.19 and 5.20 show the number of unsolved instances and CPU times and number of nodes respectively for $p_i \sim U[1,10]$. Tables 5.21 and 5.22 are the respective results for $p_i \sim U[1,100]$. It is also noted that Tables 5.19 through 5.22 are for *n*=20.

Table 5. 19 The number of unsolved instances when $p_i \sim U[1,10]$

Job (*n*)	**Scenario (*m*)**	**Set**	**Plan**	**Number of Unsolved Instances (from 10)**	
				With LB_2	**W/o LB_2**
20	3	1	1	0	0
		1	2	1	1
		2	1	0	0
		2	2	0	1
	5	1	1	0	0
		1	2	0	1
		2	1	0	0
		2	2	2	2

Table 5. 20 The CPU times and number of nodes when $p_i \sim U[1,10]$

m	**Set**	**Plan**	**Case**	**CPU Time**		**Number of Nodes**	
				Average	**Maximum**	**Average**	**Maximum**
3	1	1	With LB_2	0.033	0.042	210	211
			W/o LB_2	0.180	0.363	999	6,146
		2	With LB_2	2.263	19.315	118,571	1,061,703
			W/o LB_2	3.713	31.816	328,027	2,913,640
	2	1	With LB_2	0.220	0.404	569	3,668
			W/o LB_2	0.407	1.470	14,605	94,026
		2	With LB_2	0.407	2.483	7,686	72,036
			W/o LB_2	720.626	7200.000	47,065	398,560
5	1	1	With LB_2	0.605	3.038	7,922	39,860
			W/o LB_2	16.220	136.228	1,033,945	8,541,931
		2	With LB_2	2.666	24.904	73,820	728,869
			W/o LB_2	753.219	7200.000	2,534,953	20,599,538
	2	1	With LB_2	0.534	3.072	5,806	48,618
			W/o LB_2	2.151	10.613	147,883	849,672
		2	With LB_2	3.949	13.444	65,725	296,081
			W/o LB_2	4.513	16.039	225,608	863,181

Table 5. 21 The number of unsolved instances when $p_i \sim U[1,100]$

Job (n)	Scenario (m)	Set	Plan	Number of Unsolved Instances (from 10)	
				With LB_2	W/o LB_2
20	3	1	1	0	0
		1	2	0	0
		2	1	0	0
		2	2	0	0
	5	1	1	0	0
		1	2	3	5
		2	1	0	2
		2	2	1	1

Table 5. 22 The CPU times and number of nodes when $p_i \sim U[1,100]$

m	Set	Plan	Case	CPU Time		Number of Nodes	
				Average	Maximum	Average	Maximum
3	1	1	With LB_2	0.048	0.072	339	676
			W/o LB_2	0.567	1.994	23,327	105,171
		2	With LB_2	0.068	0.154	612	2,806
			W/o LB_2	7.870	66.669	760,375	6,664,623
	2	1	With LB_2	0.264	1.334	2,645	22,369
			W/o LB_2	1.674	8.855	70,113	520,678
		2	With LB_2	0.149	0.199	288	559
			W/o LB_2	0.334	0.529	3,705	8,961
5	1	1	With LB_2	29.001	216.975	648,147	5,040,425
			W/o LB_2	377.049	3121.483	15,746,473	113,678,551
		2	With LB_2	268.532	1677.877	1,064,552	5,130,422
			W/o LB_2	2100.014	7200.000	2,539,918	11,584,163
	2	1	With LB_2	834.931	4876.442	10,138,230	54,407,771
			W/o LB_2	1577.367	7200.000	10,521,848	52,246,240
		2	With LB_2	11.668	64.020	304,592	1,918,892
			W/o LB_2	135.151	836.731	9,997,262	57,940,943

Note from Table 5.19 that the number of unsolved instances reduces from 5 to 3 with the use of LB_2 when $p_i \sim U[1,10]$. Table 5.20 indicates that the number of nodes and the CPU times reduce drastically with the use of LB_2 when $p_i \sim U[1,10]$ For example, LB_2 reduces the CPU times from 16.220 seconds to 0.605 seconds when m is equal to 5 for set 1 and plan 1. Meanwhile, average number of nodes decreases from 1,033,945 to 7,922 with the contribution of LB_2 for the same combinations stated previously.

The similar results hold for $p_i \sim U[1,100]$. Table 5.21 reveals that LB_2 reduces the number of unsolved instances from 8 to 5. With Table 5.22, it is obvious that the CPU times and the number of nodes reduce significantly when LB_2 is used. For set 1 and plan 1, the average CPU times reduce from 0.567 to 0.048 seconds when there are 3 scenarios and from 377.049 to 29.001 seconds when there are 5 scenarios. Moreover, average number of nodes decreases from 23,327 to 339 for m=3 and 15,746,473 to 648,147 with the contribution of LB_2 for the same combinations stated previously.

The significant contribution of LB_2 is due to the difference between LB_1 and LB_2 in terms of handling the unsequenced jobs. In LB_1, the sum of processing times of unsequenced jobs are included in the bound without considering the breakdown structure. In LB_2, on the other hand, the breakdown structure is considered when computing the upper bound on the number of jobs that can be processed before the breakdown, which is then used in computing the lower bound on the makespan. Thus, solution found at the early levels of branching are close to the optimal expected makespan.

5.3.2.6 Performance of the First Feasible Solution found by the BAB

We finally investigate the performance of the first feasible solution found by the B&B. We select 60 instances as follows.

Table 5. 23 Number of instances (first feasible solution = optimal solution) for different problem settings

Number of Jobs	Problem Settings	Number of Instances (First Feasible Solution = Optimal Solution)
100	Set 1 – Plan 1	0
	Set 1 – Plan 2	0
	Set 2 – Plan 1	0
	Set 2 – Plan 2	1*
150	Set 1 – Plan 1	0
	Set 1 – Plan 2	0
	Set 2 – Plan 1	0
	Set 2 – Plan 2	1
200	Set 1 – Plan 1	0
	Set 1 – Plan 2	1
	Set 2 – Plan 1	0
	Set 2 – Plan 2	0*

*There is 1 unsolved instance out of 5 instances in this problem setting.

Table 5. 24 Deviations for instances that first feasible solution is not equal to optimal solution

Instances	Objective Value of the First Feasible Solution (OVFFS) (seconds)	Optimal Objective Value (OOV) (seconds)	Deviations ((OVFFS – OOV) / OOV)
Instance 1	5215.8	5214.0	0.030 %
Instance 2	7925.4	7925.0	0.005 %
Instance 3	9546.8	9540.0	0.070 %

We find that 58 out of 60 instances could be solved in our termination limit of 2 hours. The results have revealed that in 55 out of 58 instances, the first feasible solution turned out to be optimal. For the other 3 solved instances, the deviations of the objective value of the first feasible solution from the optimal objective value are 0.03%, 0.005% and 0.07%, i.e., the deviations are negligible.

5.3.3 Expected Value Problem and the Value of the Stochastic Solution

In this section, we first report and discuss the VSS = EEV – SP values based on the smallest-sized problem instances (n = 10), where SP values are obtained by solving the SP model.

Table 5. 25 VSS values when $p_i \sim U[1,100]$ for n=10

n	**m**	**Probability (1=Equal, 2=Later)**	**EEV-SP**	
			Average	**Maximum**
10	3	1	13.400	22.001
		2	14.864	24.003
	5	1	19.740	33.400
		2	14.747	22.883

Note from Table 5.25 that average EEV-SP values are between 13.400 and 19.740 for different combinations. Hence, when there are 10 jobs, the improvement in the expected makespan value is 15.68 on average, which indicates the benefit of using a stochastic program instead of a deterministic approach.

VSS values are calculated by taking an average of the 5 instances for the larger-sized problem instances (n = 50 and n = 100) and are reported in Tables 5.26 and 5.27. Note that, SP values are obtained by using the proposed B&B algorithm for these instances. For EVP values, we utilize the same approach used for the smaller-sized instances, which is described in Part 3.5. Remember that we can further simplify the EVP by eliminating the sequencing decisions, and this would result in

a knapsack problem. After determining which jobs are processed before the expected breakdown by solving this model, SPT is used to sequence the jobs in each group, and the expected makespan is computed for the generated sequence.

Table 5. 26 VSS values for n=50, m=5 and $p_i \sim U[1,100]$

Problem Settings	**Average value of VSS** $(\boldsymbol{VSS = z_{EEV} - z^*_{SP}})$ **(seconds)**
Set 1 - Plan 1	12.76
Set 1 - Plan 2	22.12
Set 2 - Plan 1	22.31
Set 2 - Plan 2	21.60

Table 5. 27 VSS values for n=100, m=5 and $p_i \sim U[1,100]$

Problem Settings	**Average value of VSS** $(\boldsymbol{VSS = z_{EEV} - z^*_{SP}})$ **(seconds)**
Set 1 - Plan 1	21.20
Set 1 - Plan 2	23.84
Set 2 - Plan 1	23.56
Set 2 - Plan 2*	34.88

* 4 instances are considered as one instance is not solved in the termination limit.

VSS represents the expected improvement in the objective function as a result of solving the stochastic model (using the B&B algorithm) instead of a deterministic model, the knapsack model for this case. Improvement values range between 12.76 and 22.31 for n=50 and between 21.20 and 34.88 for n=100. It can be concluded that the value of solving a stochastic model rather than a deterministic one becomes higher as the number of jobs (n) increases.

Table 5. 28 VSS values for n=50, m=3 and $p_i \sim U[1,100]$

Problem Settings	**Average value of VSS** $(\boldsymbol{VSS = z_{EEV} - z^*_{SP}})$ **(seconds)**
Set 1 - Plan 1	6.47
Set 1 - Plan 2	21.06
Set 2 - Plan 1	20.06
Set 2 - Plan 2	20.07

We also investigated the effect of the number of scenarios (m) on the VSS values. We report the VSS values for n = 50 and $m = 3$ (i.e., for a smaller number of scenarios) in Table 5.28. When the values in Tables 5.26 and 5.28 are compared, it is observed that VSS is higher when more scenarios are used to capture the uncertainty.

Table 5. 29 VSS values for n=50, m=5 and $p_i \sim U[1,10]$

Problem Settings	**Average value of VSS** $(\boldsymbol{VSS = z_{EEV} - z^*_{SP}})$ **(seconds)**
Set 1 - Plan 1	1.60
Set 1 - Plan 2	2.28
Set 2 - Plan 1	2.43
Set 2 - Plan 2	2.07

To observe the effect of processing times on the VSS values, we executed experiments for n=50, m=5 and $p_i \sim U[1,10]$. We report the VSS values for this set of experiments in Table 5.29. From the comparison of the values in Tables 5.26 and 5.29, we can conclude that the instances with $p_i \sim U[1,10]$ produce very close

values for different job sequences due to the similarity of the processing times. Therefore, the VSS values are higher for $p_i \sim U[1,100]$ setting.

CHAPTER 6

CONCLUSION

In this study, we consider a single machine scheduling problem with random breakdowns. We assume that there is a single breakdown with an unknown occurrence time. The occurrence times are stochastically known with probabilities. Each occurrence time is referred to as a scenario. We aim to find a sequence of jobs to maximize the expected makespan. We propose two stochastic programming models one of which takes precedence over the jobs and the other takes the job position as the main decision.

To assess the value of using a stochastic programming approach, we find the expected value solution and evaluate its expected makespan value. The expected value solution is found through a single knapsack model. We observe that there is difference between the optimal objective function value of a stochastic program and the expected makespan of the expected solution, which justifies the use of a stochastic program.

Our experiments have revealed that the precedence based model performs superior to the position model, and it can solve instances with up to 10 jobs.

To handle the medium to large sized problem instances, we propose a branch and bound algorithm. We provide an efficient branching scheme that avoids the repetition of many partial solutions. We improve the performance of the branch and bound algorithm with two powerful lower bounding approaches. Our lower bounds find a lower bound for each scenario realization and weigh the scenario lower bounds by their occurrence probabilities.

Our extensive computational results with up to 200 jobs and 5 scenarios have revealed the satisfactory behavior of the branch and bound algorithm. We observe a more dominant effect of the number of scenarios than the number of jobs, since its impact is not only on the problem size but also on the length of the period before the breakdown. The exponential nature of the search is dispelled significantly through our efficient branching scheme and lower bounding schemes. Most of the instances would remain unsolved for two hours in the absence of those mechanisms. We observe the significance of the maintenance patterns and processing time distributions on the performance of the algorithm. The instances with periodic maintenance and low processing time variability are the easiest to solve.

To the best of our knowledge, we propose the first stochastic programming approach to a single machine problem with uncertain breakdown times. We hope that our promising results may trigger developments in the stochastic scheduling area. Future research may consider the development of B&B based heuristic approaches like filtered beam search algorithm that uses our branching scheme and lower bounds. An extension of our study to more complex objective functions like total flow time and total tardiness is another fruitful research area. Moreover, more complex breakdown structures, like multiple breakdowns, are worth-studying.

REFERENCES

A comparative study on stochastic programming problems: Modeling ... (n.d.). Retrieved August 21, 2022, from https://www.researchgate.net/publication/338083207_ A_ comparative_study_on_Stochastic_Programming_Problems_modeling_and_applications_in_real_life

Abdul Halim, N. N., Shariff, S. S., & Zahari, S. M. (2020). Single-Machine Integrated Production Preventive Maintenance Scheduling: A Simheuristic approach. *MATEMATIKA*, *36*(2), 113–126.

Al-Khamis, T., & M'Hallah, R. (2011). A two-stage stochastic programming model for the parallel machine scheduling problem with machine capacity. *Computers & Operations Research, 38*(12), 1747–1759.

Atakan, S., Bülbül, K., & Noyan, N. (2016). Minimizing value-at-risk in single machine scheduling. *Annals of Operations Research*, *248*(1-2), 25–73.

Batun, S., & Azizoğlu, M. (2009). Single Machine scheduling with preventive maintenances. *International Journal of Production Research, 47*(7), 1753–1771.

Birge, J.R., and Louveaux, F., (2011), "Introduction to Stochastic Programming". Springer, New York.

Cassady, C. R., & Kutanoglu, E. (2005). Integrating Preventive Maintenance Planning and production scheduling for a single machine. *IEEE Transactions on Reliability, 54*(2), 304–309.

Chen, W. (2009). Minimizing number of tardy jobs on a single machine subject to periodic maintenance. *Omega*, *37*(3), 591–599.

Chen, Y., Huang, C., Chou, F. D., & Huang, S. (2020). Single-machine scheduling problem with flexible maintenance and non-resumable jobs to minimise makespan. *IET Collaborative Intelligent Manufacturing*, *2*(4), 174–181.

Escudero, L. F., Garín, A., Merino, M., & Pérez, G. (2007). The value of the stochastic solution in multistage problems. *TOP*, *15*(1), 48–64.

Hariga, M. (1994). A deterministic maintenance-scheduling problem for a group ofnon-identical machines. *International Journal of Operations & Production Management*, *14*(7), 27–36.

Liu, M., Wang, S., Chu, C., & Chu, F. (2015). An improved exact algorithm for single-machine scheduling to minimise the number of tardy jobs with periodic maintenance. *International Journal of Production Research*, *54*(12), 3591–3602.

Low, C., Ji, M., Hsu, C.-J., & Su, C.-T. (2010). Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintenance. *Applied Mathematical Modelling*, *34*(2), 334–342.

Ozcelik, F., Ertem, M., & Saraç, T. (2021). A stochastic approach for the single-machine scheduling problem to minimize total expected cost with client-dependent tardiness costs. *Engineering Optimization*, *54*(7), 1178–1192.

Pan, E., Liao, W., & Xi, L. (2010). Single-machine-based production scheduling model Integrated Preventive Maintenance Planning. *The International Journal of Advanced Manufacturing Technology*, *50*(1-4), 365–375.

van den Akker, M., & Hoogeveen, H. (2007). Minimizing the number of late jobs in a stochastic setting using a chance constraint. *Journal of Scheduling*, *11*(1), 59–69.

van den Akker, M., Hoogeveen, H., & Stoef, J. (2018). Combining two-stage stochastic programming and recoverable robustness to minimize the number of late jobs in the case of Uncertain Processing Times. *Journal of Scheduling*, *21*(6), 607–617.

Wang, S., & Liu, M. (2013). A branch and bound algorithm for single-machine production scheduling integrated with Preventive Maintenance Planning. *International Journal of Production Research*, *51*(3), 847–868.