PH.D. THESIS

Simulation and Optimization of Scheduling Policies in Dynamic Stochastic Resource-Constrained Multi-Project Environments

Author:
Ugur Satic

Supervisors:
Dr Peter Jacko and Dr Christopher Kirkbride

A Thesis Submitted in Partial Fulfillment for the Degree of Doctor of Philosophy

Department of Management Science
Lancaster University

Lancaster, The United Kingdom
January 2022

Declaration

I confirm that the work presented in this thesis is based on the original research which has been carried out by me under the supervision of my supervisors Dr Peter Jacko and Dr Christopher Kirkbride. This thesis has not been previously submitted elsewhere for the award of any other degree. The information derived from other sources has been acknowledged in the text, and a list of references is provided.

Chapter 4 which is published as "Performance evaluation of scheduling policies for the DRCMPSP" at Analytical and Stochastic Modelling Techniques and Applications. ASMTA 2019. Lecture Notes in Computer Science, vol 12023 (Satic et al., 2020a), Chapter 5 which is published as "Performance evaluation of scheduling policies for the Dynamic and Stochastic Resource-Constrained Multi-Project Scheduling Problem" at International Journal of Production Research (Satic et al., 2020b), Chapter 6 which is submitted as "A Simulation-Based Approximate Dynamic Programming Approach to Dynamic and Stochastic Resource Constrained Multi-Project Scheduling Problem" to European Journal of Operational Research are joint works done by myself and my supervisors Dr Peter Jacko and Dr Christopher Kirkbride. The other chapters are original works done by myself under the supervision of Dr Peter Jacko and Dr Christopher Kirkbride.

Ugur SATIC
January 2022

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisors Dr Peter Jacko and Dr Christopher Kirkbride, for giving me the opportunity to do research at the Department of Management Science at Lancaster University Management School. This thesis could not have been possible without the guidance and expertise of my supervisors. I am very thankful to Dr Peter Jacko and Dr Christopher Kirkbride for all their help and supervision since the preparation of my thesis proposal.

I am thankful to PhD programmes coordinator Mrs Gay Bentinck for providing support and guidance.

This PhD study has been funded by the Ministry of National Education of The Republic of Turkey. I am grateful to my sponsor for their funding which made this PhD study possible.

I am grateful to Mahshid Salemi Parizi for their work Parizi et al. (2017) and for making their code available. Their work helped me understand the application of an approximate dynamic programming method on the dynamic resource-constrained project scheduling problem. Moreover, it inspired me to create my approximate dynamic programming model on Chapter 6.

Simulation and Optimization of Scheduling Policies in Dynamic Stochastic Resource-Constrained Multi-Project Environments

Ugur Satic, M.Sc.
Department of Management Science, Lancaster University
A Thesis Submitted in Partial Fulfillment for the Degree of Doctor of Philosophy
January 2022

Abstract

The goal of the Project Management is to organise project schedules to complete projects before their completion dates, specified in their contract. When a project is beyond its completion date, organisations may lose the rewards from project completion as well as their organisational prestige. Project Management involves many uncertain factors such as unknown new project arrival dates and unreliable task duration predictions, which may affect project schedules that lead to delivery overruns. Successful Project Management could be done by considering these uncertainties. In this PhD study, we aim to create a more comprehensive model which considers a system where projects (of multiple types) arrive at random to the resource-constrained environment for which rewards for project delivery are impacted by fees for late project completion and tasks may complete sooner or later than expected task duration.

In this thesis, we considered two extensions of the resource-constrained multi-project scheduling problem (RCMPSP) in dynamic environments. RCMPSP requires scheduling tasks of multiple projects simultaneously using a pool of limited renewable resources, and its goal usually is the shortest make-span or the highest profit. The first extension of RCMPSP is the dynamic resource-constrained multi-project scheduling problem. Dynamic in this problem refers that new projects arrive randomly during the ongoing project execution, which disturbs the existing project scheduling plan. The second extension of RCMPSP is the dynamic and stochastic resource-constrained multi-project scheduling problem. Dynamic and stochastic represent that both random new projects arrivals and stochastic task durations. In these problems, we assumed that projects generate rewards at their completion; completions later than a due date cause tardiness costs, and we seek to maximise average profits per unit time or the expected discounted long-run profit. We model these problems as infinite-horizon discrete-time Markov decision processes.

Extended Abstract

The goal of the Project Management is to organise project schedules to complete projects before their completion dates, specified in their contract. When a project is beyond its completion date, organisations may lose the rewards from project completion as well as their organisational prestige. Project Management involves many uncertain factors such as unknown new project arrival dates and unreliable task duration predictions, which may affect project schedules that lead to delivery overruns. Successful Project Management could be done by considering these uncertainties. In this PhD study, we aim to create a more comprehensive model which considers a system where projects (of multiple types) arrive at random to the resource-constrained environment for which rewards for project delivery are impacted by fees for late project completion and tasks may complete sooner or later than expected task duration.

In this thesis, we considered two extensions of the resource-constrained multi-project scheduling problem (RCMPSP) in dynamic environments. RCMPSP requires scheduling tasks of multiple projects simultaneously using a pool of limited renewable resources, and its goal usually is the shortest make-span or the highest profit. The first extension of RCMPSP is the dynamic resource-constrained multi-project scheduling problem. Dynamic in this problem refers that new projects arrive randomly during the ongoing project execution, which disturbs the existing project scheduling plan. The second extension of RCMPSP is the dynamic and stochastic resource-constrained multi-project scheduling problem. Dynamic and stochastic represent that both random new projects arrivals and stochastic task durations. In these problems, we assumed that projects generate rewards at their completion; completions later than a due date cause tardiness costs, and we seek to maximise average profits per unit time or the expected discounted long-run profit. We model these problems as infinite-horizon discrete-time Markov decision processes.

In Chapter 4, we consider the dynamic resource-constrained multi-project scheduling problem and explore the computational limitations of solving the problem by dynamic programming. We run and compare four different solution

approaches on small size problems. These solution approaches are: a dynamic programming algorithm to determine a policy that maximises the average profit per unit time net of charges for late project completion, a genetic algorithm which generates a schedule to maximise the total reward of ongoing projects and updates the schedule with each new project arrival, a rule-based algorithm which prioritises processing of tasks with the highest processing durations, and a worst decision algorithm to seek a non-idling policy to minimise the average profit per unit time. Average profits per unit time of generated policies of the solution algorithms are evaluated and compared. The performance of the genetic algorithm is the closest to the optimal policies of the dynamic programming algorithm, but its results are notably suboptimal, up to 67.2%. Alternative scheduling algorithms are close to optimal with low project arrival probability but quickly deteriorate their performance as the probability increases. In Chapter 5, we consider the dynamic and stochastic resource-constrained multi-project scheduling problem and explore the computational limitations of solving it by dynamic programming. We run and compare five different solution approaches on small size problems, which are generated by including stochastic task durations to problems from Chapter 4. Four of the solution approaches are from Chapter 4 and the fifth is an optimal reactive baseline algorithm that generates the best schedule to maximise the total profit of ongoing projects (without considering the new arrivals or the stochastic task completion). The performance of the optimal reactive baseline algorithm is the closest to the optimal policies of the dynamic programming algorithm, but its results are suboptimal, up to 37.6%. In Chapter 6, we consider the dynamic and stochastic resource-constrained multi-project scheduling problem, which are larger than the computational limitations of solving it by dynamic programming. On top of the features considered in Chapter 5, we also consider non-sequential project networks, availability of multiple resource types and allocation of multiple resource types for the same task of a project. We use an approximate dynamic programming algorithm with a linear approximation model, which can be used for online decision-making. Our approximation model uses project elements that are easily accessible by a decision-maker with the model coefficients obtained offline via a combination of Monte Carlo simulation and the least-squares estimation method. We run and compare our approximate dynamic programming algorithm along with the solution approaches from Chapter 5. Our numerical study shows that approximate dynamic programming and optimal reactive baseline algorithm produce similar results, which are typically both inferior to the optimal results of dynamic programming. Approximate dynamic programming has an advantage over optimal reactive baseline algorithm and dynamic programming in that

approximate dynamic programming could be applied to larger problems. We also show that approximate dynamic programming generally produces statistically significantly higher profits than common algorithms used in practice, such as a rule-based algorithm and a reactive genetic algorithm.

In Chapter 1, the introduction, contributions of this thesis, thesis structure and status of publications is presented. In Chapter 2, we summarise the literature of the dynamic resource-constrained multi-project scheduling, the dynamic and stochastic resource-constrained multi-project scheduling. In Chapter 3, we explain the methods used in this thesis. In Chapter 7, the conclusion is presented.

Contents

List of Figures

Chapter 1

Introduction

"A *project* is a unique, transient endeavour, undertaken to achieve planned objectives, which could be defined in terms of outputs, outcomes or benefits" (APM, 2012). Uniqueness and time-bound are the main features of a project which differentiates it from a regular working activity. Coordinating the limited *resources* (e.g. human power, equipment) between projects to get the most benefit from outcomes is called *project management*.

Project management is a crucial part of most businesses because it provides plans to manage projects. However, project management is a very challenging enterprise in that only 40% of projects are completed within their planned time, 46% of projects are completed within their predicted budget, and 36% of projects realise their full benefits (Wellingtone PPM, 2018).

Project management involves high risk and uncertainty. In many sectors such as engineering services, make-to-order services, software development, IT services, construction and R&D, project management involves multiple uncertain elements; for example, unpredicted new project arrivals and unreliable task duration predictions, which are frequently ignored by traditional project management studies.

Project management seeks to maximise project outcome and benefit while minimising the risks and costs. A *reward* is released as the outcome of project completion. However, due to the time-bounded feature of projects, completion of the project beyond a pre-determined *due date* may cause penalties which are called the *tardiness cost*. The software development project is an excellent example of this project definition. Alhumrani and Qureshi (2016) state that software development projects involve the processing of multiple projects at the same time with shared resources. Furthermore, these projects have strict due dates that, after this date, the customer may not accept these projects and all spent resources and time may be lost.

The motivation of this study is to create a more comprehensive project scheduling model by considering the uncertainties of stochastic task durations and new project arrivals. The research outcome will be valuable to project managers to plan their work and calculate costs accurately.

1.1 Thesis contributions

In this thesis, we model the problem as an infinite-horizon discrete-time Markov decision process (MDP) where decisions about task processing are made in fixed intervals. The period between two decision is called transition time and used as a time unit. The time unit can be minutes, hours, days or weeks depending on the problem. During a transition time, the system processes projects according to taken decisions, and random changes occur. The random arrival of new projects and stochastic completion of tasks are considered as random change in Chapter 5 and Chapter 6. In Chapter 4, only the random arrival of new projects is considered.

We contribute to the literature by (i) developing exact models of the dynamic resource-constrained multi-project scheduling problem (dynamic RCMPSP) and the dynamic and stochastic resource-constrained multi-project scheduling problem (dynamic and stochastic RCMPSP) which extends the work of Melchiors et al. (2018), who only considered single-task projects, by considering multi-task project types, (ii) developing a new comprehensive MDP model which considers the random arrival of new projects, stochastic task durations, multiple resource types, non-sequential project networks, project due dates and tardiness costs, (iii) comparing the optimal policy obtained by value iteration with optimal reactive baseline algorithm, genetic algorithm, approximate dynamic programming algorithm and rule based algorithm to evaluate the performance gap between solution approaches, (iv) illustrating that even in simple problems with 2 or 3 project types, the suboptimality gap of benchmark policies commonly used in practice (genetic algorithm and longest-task-first rule) which ignore possibility of new project arrivals is remarkable. (v) developing a new Approximate Dynamic Programming (ADP) approximation function that uses project rewards, tardiness costs, spent processing time and resource usage of projects for decision making, and it is capable of solving much larger, more complex and much more general problems than ADPs from existing literature. (vi) efficiency of our ADP approach. In 13.3% of compared problems, the results of our ADP approach are not statistically significantly different than optimum results of DP. Compare to popular methods in the RCMPSP literature, our ADP approach generated statistically significantly higher results than GA and RBA respectively in 57% and 69% of the compared

problems. The results of our ADP approach are statistically significantly lower than GA and RBA respectively only in 13% and 2% of the compared problems.

1.2 Thesis structure

This thesis is organised into seven chapters. The next chapter is Chapter 2, where we summarise the literature of the dynamic RCMPSP and the dynamic and stochastic RCMPSP. Also, we will mention the origin of our considered problems such as RCPSP, RCMPSP and their stochastic equivalents, which are the stochastic RCPSP and stochastic RCMPSP.

In Chapter 3, the methods which are used in this thesis are explained. These methods are dynamic programming (DP), genetic algorithm (GA), rule-based algorithm (RBA), worst decision algorithm (WDA), optimal reactive baseline algorithm (ORBA) and approximate dynamic programming (ADP).

In Chapter 4, we consider the dynamic RCMPSP where projects generate rewards at their completion, completions later than a due date cause tardiness costs, and new projects arrive randomly during the ongoing project execution which disturb the existing project scheduling plan. We model this problem as a discrete-time Markov decision process (MDP). We explore the performance and computational limitations of solving the problem by DP with the goal of maximising the time-average profit. GA and RBA are the most used (or common) algorithms in RCPSPs. We compare the results of DP with a GA which generates a schedule to maximise the total profit of ongoing projects and a priority RBA which prioritises processing of tasks with the highest processing durations. We also use a WDA, which seek a non-idling policy that minimises the time-average profit, to make a comparison with the lowest result can be made. Due to the limitation of DP, we simplify the problem by considering only sequential project networks, single project arrival at each transition time, a single type of resource, non-preemptive task processing and a small number of project types with a small number of tasks.

In Chapter 5, we consider the dynamic and stochastic RCMPSP which is an extension of the dynamic RCMPSP with stochastic durations of tasks. We create test problems by adding early and late completions probabilities to problems from Chapter 4. We use the deterministic task durations of problems from Chapter 4 as the expected task duration. In addition to DP, GA and RBA used in Chapter 4, we introduce an ORBA that generates an optimal schedule for the static problem (i.e., assumes no new arrivals in the future). Due to the limitation of DP, we use all assumptions from Chapter 4. For stochastic task durations, we assume that tasks may complete only one period early or a period later than their expected

processing times.

In Chapter 6, we extended our work in Chapter 5 by considering multiple resource types and non-sequential project networks. We also considered bigger size (more project types and more task number) problems than we consider in Chapter 5. We model this problem as a discrete-time Markov decision process (MDP), and we seek to maximise the discounted long time profit. We use an ADP algorithm with a linear approximation model to approximate the value function of Bellman's equation. Our approximation model uses resource consumption and processing times of project types as features. This model can be used for online decision-making after estimating coefficients of the linear value function approximation using a simulation-based training phase. We compare the performance of our ADP algorithm with solution methods which are used in Chapter 5. We used the problems from Chapter 5 for comparison. Also, we generate new comparison problems that are larger and include non-sequential project networks and multiple resource types. The larger size problems are computationally intractable for DP and ORBA; thus, we benchmark ADP with GA and RBA on these problems. In Chapter 6, we assume that only a single project of each type may arrive at each transition time, tasks may complete only one period early or a period later than their expected processing times and task processing is non-preemptive.

In Chapter 7, the conclusion is presented.

1.3 Status of publications

Chapter 4 was presented at "The OR Society's 63rd Annual Conference (OR60)" on September 13, 2018 in Lancaster, and it was presented again at "The 25th International Conference on Analytical & Stochastic Modelling Techniques & Applications (ASMTA-2019)" on October 24, 2019 in Moscow. This work is published as "Performance evaluation of scheduling policies for the DRCMPSP" in Analytical and Stochastic Modelling Techniques and Applications. ASMTA 2019. Lecture Notes in Computer Science, vol 12023. Chapter 4 is accessible online as https://doi.org/10.1007/978-3-030-62885-7_8

Chapter 5 was presented at "The 30th European Conference on Operational Research (EURO2019)" on June 25, 2019 in Dublin and published as "Performance evaluation of scheduling policies for the Dynamic and Stochastic Resource-Constrained Multi-Project Scheduling Problem" in International Journal of Production Research. Chapter 5 is accessible online as https://doi.org/10.1080/00207543.2020.1857450

Chapter 6 was presented at "The OR Society's 63rd Annual Conference (OR63)" on September 14, 2021 and submitted as "A Simulation-Based Approximate Dynamic Programming Approach to Dynamic and Stochastic Resource Constrained Multi-Project Scheduling Problem" to European Journal of Operational Research.

Chapter 2

Project scheduling literature

Dynamic and stochastic RCMPSP is an extension of the *resource-constrained project scheduling problem* (RCPSP). This chapter will review studies in the field of RCPSP and its extensions which are shown in Figure 2.1. First, the general RCPSP problem and its multi-project version *resource-constrained multi-project scheduling problem* (RCMPSP) will be mentioned. Then their stochastic extensions, the *stochastic resource-constrained project scheduling problem* (stochastic RCPSP) and the *stochastic resource-constrained multi-project scheduling problem* (stochastic RCMPSP) will be reviewed. After that, we will show the literature summary of the dynamic RCMPSP. Finally, dynamic and stochastic RCMPSP will be discussed in more detail.

2.1 Project scheduling problem (PSP) and resource-constrained project scheduling problem (RCPSP)

The project scheduling problem (PSP) is an NP-hard class problem, and its goals are optimisation of project duration, resource allocation, cost evaluation or cash flow (Ortiz-Pimiento and Diaz-Serna, 2018). NP-hard refers to the fact that these problems can not be solved or verified by exact algorithms in polynomial time. In PSP, the optimisation goals are accomplished by finding the optimal start times of tasks or the optimal sequence of tasks. Most PSP can be solved by well-known network-based project management techniques, e.g., the critical path method (CPM) and the project management and review technique (PERT).

PSP has many extensions. The PSP with at least one type of resource is constrained by an amount is called resource-constrained PSP (RCPSP) (Kolisch and Padman, 2001). The goal of RCPSP is determining the best order of the tasks that maximise project profit (i.e. reward, less tardiness cost) or minimise

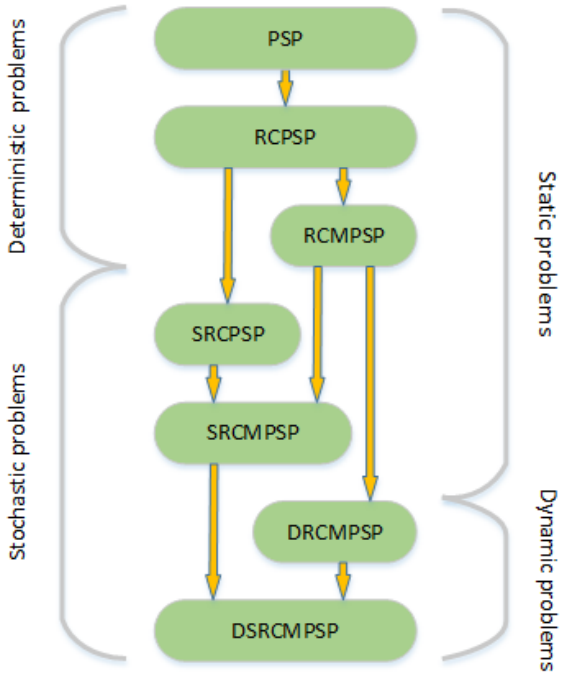

Figure 2.1: A flowchart of RCPSP and its extensions

the project completion time by utilising the limited pool of available resources. RCPSP is extensively studied by a large number of exact algorithms and heuristic methods: priority rule-based methods, genetic algorithm, ant colony optimization, simulated annealing, particle swarm optimization, tabu search, neural network, scatter search, neighbourhood search and hybrid forms of these methods are some of the applied solution methods in the literature (Karam and Lazarova-Molnar, 2013).

Karam and Lazarova-Molnar (2013) cites that small-sized RCPSPs can be solved by exact algorithms within acceptable time but the required computation time for large-sized RCPSPs is impractical. Thus, usually heuristics methods are used for large-sized RCPSPs. It also states that the common goal of the RCPSP is minimising the completion time, and *genetic algorithm* is the most used solution algorithm. Well-known RCPSP test problems are available at PSPLIB (Kolisch and Sprecher, 1997), which is an online RCPSP library (http://www.om-db.wi.tum.de/psplib).

The RCPSP is one of the most extensively studied research problem (Creemers, 2015). As stated above, many solution approaches have been applied to the RCPSP. However, it is not the main topic of this thesis, and it is not the purpose of this study to provide an exhaustive literature review. Thus here, we highlight a few applications of solution methods used for these problems. Kolisch and Drexl (1996) proposed an adaptive search procedure using priority rules and random search techniques for the RCPSP. The objective function is the minimisation of

the project makespan via minimising the completion times of each task. The algorithm is tested using the benchmark set of Patterson (1984). Hartmann (1998) proposed a self-adapting genetic algorithm for the RCPSP. They created permutation of schedules and compared their performance with a priority rule-based genetic algorithm using the PSPLIB problems. Hartmann (2002) proposed another self-adapting genetic algorithm for the RCPSP. This model differs from Hartmann (1998) by having both serial and parallel schedule generation scheme (SGS). Khamooshi (1999) claimed that choosing one priority rule during the whole RCPSP might let a sub-optimal solution. Thus they offered to divide a project into periods and choose the best priority rule for each period to get a better solution. This approach is called the dynamic priority scheduling method. They used a DP model to select the best priority rules for each period. They tested the algorithm over one hundred problems and outperformed the single period approach on almost all of the problems.

2.2 Resource-constrained multi-project scheduling problem (RCMPSP)

In the current economic environment, companies manage multiple projects from different customers at the same time (Homberger, 2012). Thus simultaneous management of multiple projects has become very common and crucial (Browning and Yassine, 2010). The RCPSP, which extended with more than one project, is called *resource-constrained multi-project scheduling problem* (RCMPSP). Two RCMPSP solution approaches exist: (1) the first approach combines projects in parallel with a dummy start-task and a dummy end-task, then solves the problem as a giant project network, (2) the second approach maintains the multiple projects networks separately (Browning and Yassine, 2010). The general goal of the RCMPSP is minimising the total (for the first approach) or the average (for the second approach) completion time (Browning and Yassine, 2010). The first approach is more common in the literature and most algorithms developed for RCPSP can be applied to RCMPSP with this approach.

An RCMPSP library named MPSPLIB is available at "http://www.mpsplib.com/" which contains sets of problems generated by Homberger (2012). We used some problems from MPSPLIB in our comparison in subsection 6.5.3. Here we highlight that some researchers used MPSPLIB problems in their comparison. Homberger (2012) has developed a multi-agent system algorithm for the RCMPSP. The model uses a coordination mechanism based on an evolutionary search. They used multi-project test problems, which are combinations of the PSPLIB problems

and compared the performance of their algorithm with Homberger (2007). The test problems of this research are available at MPSPLIB. Adhau et al. (2012) proposed a distributed multi-agent algorithm for the RCMPSP, using auction-based negotiation. Their algorithm seeks the optimum project delays and total project completion without resource conflicts. Adhau et al. (2013) developed this algorithm by considering the cost and duration of moving resource between projects. The new algorithm aimed to minimise the duration and resource costs in addition to project delays and total project completion. However, moving resource is not considered in their comparison chapter to able to solve MPSPLIB problems.

2.3 Stochastic RCPSP and stochastic RCMPSP

Despite the vast number of studies in the field of the RCPSP and RCMPSP, some researchers (e.g., Creemers (2015) and Chand et al. (2019)) claimed that these deterministic problems rarely represent real-life problems that involve uncertainties and considering a deterministic environment is not realistic. In the literature, the generalisations of RCPSP and RMCPSP with uncertainty in project features or resources are called the *stochastic* RCPSP (SRCPSP) and the *stochastic* RCMPSP (SRCMPSP) respectively.

The common goal of these stochastic problems is to minimize the *expected* completion time (Rostami et al., 2018). Ortiz-Pimiento and Diaz-Serna (2018) showed that PSP with stochastic task durations have drawn great attention since 2007, stating that approximately half of these papers focused on the stochastic RCPSP and the stochastic RCMPSP, and the most used solution algorithm for these problems is genetic algorithm. In the majority of research in the stochastic RCPSP and the stochastic RCMPSP areas, the uncertainty of stochastic task duration is considered (e.g. Tereso et al. (2004); Deblaere et al. (2011); Creemers (2015); Li and Womer (2015); Yassine et al. (2017); Wang et al. (2017); Bruni et al. (2018); Rostami et al. (2018)). However, some (e.g. Wang et al. (2015); Song et al. (2018); Chand et al. (2019)) consider the stochastic resource availability.

The literature review of Ortiz-Pimiento and Diaz-Serna (2018) shows that meta-heuristic methods such as genetic algorithm, particle swarm optimisation, tabu search, bee colony, ant colony, greedy algorithms, simulated annealing and distribution estimation algorithm; exact methods such as branch and bound, dynamic programming and stochastic programming; special procedures such as priority rules-based, simulation process-based or stage-by-stage analysis based; critical chain methods are some of the applied solution methods in the literature. Also, an approximate dynamic programming method is used for stochastic RCPSP

by Li et al. (2020). Recent examples of the application of such methods are now described below:

Tereso et al. (2004) considered the stochastic RCPSP with exponentially distributed multi mode task durations. Their problem have four optional task durations with equal probabilities and one resource type. First they used a PERT method with expected task durations to find the critical pathway of the project network. Using the critical pathway, they used fixed resource usages for un-critical tasks. Then they solved the problem with dynamic programming with the goal of minimising the resource allocation and tardiness costs. Finally, they optimised the un-critical tasks after calculating the expected costs.

Creemers (2015) modelled the stochastic RCPSP as a continuous-time MDP and used acyclic phase-type distribution for the task durations. The phase-type distribution is consist of multiple exponentially distributed building blocks. They offered a stochastic dynamic programming model by extending the study of Creemers et al. (2010). Their algorithm is capable of solving small to medium size (up to sixty tasks) problems.

Wang et al. (2015) modelled the stochastic RCMPSP as a discrete-time MDP where the time unit was a day. They used deterministic task durations to focus on the uncertainty of resource availability by random events such as resource failures and resource allocations of out-of-schedule urgent projects. These two events affect the available number of resources. They proposed a dynamic programming algorithm with strategy approximation. The goal of the algorithm is minimising the expected total tardiness penalties. Five priority rules, which are; weigh the shortest processing time (WSPT), apparent tardiness cost (ATC), earliest due date (EDD), first-in, first-out (FIFO), and cost over time, are used to limit the action and state spaces. They stored the system time, numbers of available resources, start times of the tasks in the state spaces.

Deblaere et al. (2011) considered the stochastic RCPSP and formulated it as a news-vendor problem with early and late task completion costs. Their task durations have beta distribution. So they used medians of simulated task durations. They used a heuristic algorithm to create the initial policy. Then they performed neighbourhood search to improve the initial policy and evaluated it with simulation.

Yassine et al. (2017) offered two genetic algorithm approaches for the stochastic RCMPSP with iteration and triangularly (minimum, most likely, maximum) distributed task durations. The iteration represents repeating a task processing according to new feedback such as testing and task integration. They used repeating probabilities to repeat a task and decreased a task duration after its

repeat. This study is the first research that compares the performances of a genetic algorithm and priority rules on this problem.

Wang et al. (2017) compared twenty priority rules taken from Browning and Yassine (2010) on the stochastic RCMPSP with triangularly distributes task durations. They categorised the priority rules based on solution quality and robustness using test scenarios that have different settings. The solution quality is the duration difference between a solution and the critical pathway, and the robustness was the consistency of a solution's performance between different scenarios. They created the test samples using a project scheduling problem generator (RanGen2) with uniformly distributed task durations and resource requirements.

Rostami et al. (2018) used a two-phase meta-heuristic for the stochastic RCPSP. Their first phase is a greedy randomized adaptive search procedure (GRASP), and their second phase is a genetic algorithm. They used expected task durations using simulation.

Chand et al. (2019) focused on the stochastic RCPSP where resource availability varies with time. They considered random resource disruptions which affect the resource availability and used pre-emptive task duration during the disruptions. They proposed a genetic programming hyper-heuristic to evolve priority heuristics and tested their algorithm with scenarios that are created by the combination of PSPLIB problems and a disruption generator.

Li and Womer (2015) modelled the stochastic RCPSP as a continues-time MDP where task duration are exponentially distributed. They used a rollout policy approximate dynamic programming algorithm and a look-up table to reduce Monte-Carlo simulation samples. First they generated a look-up table which has state and action pairs with expected task durations by Monte-Carlo simulation. For a state transition, they finds the action set. then, for each actions, they check the look-up table for task durations. If action is not available in the look-up table, They use a heuristic algorithm which creates the task durations with Monte-Carlo simulation.

Bruni et al. (2018) designed the stochastic RCPSP as a two-stage stochastic mixed-integer model where task durations discretely distributed. They suggested a stochastic programming optimisation approach based on the L-Shaped Method for the problem. The algorithm decomposes the original problem into a master problem and some sub-problems. The sub-problems are scenarios of task duration realisations, and the master problem generates a sub-optimal solution based on the scenarios. They generates a master and sub-problems until an optimal schedule is found, which minimises the expected make-span over all created scenarios.

Song et al. (2018) considered the uncertain factors, which could affect the workability, and uncertain failures, which may lead to repeating a task. They proposed an agent-based simulation system for the stochastic RCMPSP. The algorithm aims to minimise the the make-span by generating a baseline schedule. Then it modifies this schedule according to simulation results.

2.4 Dynamic RCMPSP

The RCMPSP assumes that the arrival times of the projects are known before the project plan. In reality, it is not usually possible to predict the arrival times of new projects. The new project arrival is an uncertain event. These uncertain events deviate from the project plan and lead to missed due dates and associated tardiness costs (Capa and Ulusoy, 2015).

Many companies accept new projects during the processing of ongoing projects (Herbots et al., 2007). However, it might not be possible to complete the new projects before their due date if their processing begins after completion of the ongoing project schedule. Thus, new projects should be added to execution as soon as possible without waiting for completion of the ongoing schedule. So the project plan should be updated when a new project has arrived. We address the dynamic RCMPSP in Chapter 4.

The uncertainty of project arrivals requires considering the RCMPSP as a dynamic environment. In this dynamic environment, the RCMPSP with uncertain project arrivals is called *dynamic* RCMPSP (DRCMPSP). The goal of dynamic RCMPSP is to find optimal schedules or scheduling policies that maximise some function of *profit*, which is the difference between the completion rewards and tardiness costs. In general, there are three standard objective functions for dynamic problems: (i) the expected total profit over a finite horizon, (ii) the expected total discounted profit over a finite or infinite horizon (e.g., Parizi et al. (2017)), or (iii) the expected time-average profit over an infinite horizon (e.g., Wang et al. (2015)).

The non-dynamic variants of RCMPSP are extensively studied (Creemers, 2015). However, the dynamic variants of the RCMPSP where new projects randomly arrive in the system are scarce in the literature. To the best of our knowledge, there are only three research papers available for the dynamic RCMPSP, which are Pamay et al. (2014), Parizi et al. (2017) and Chapter 4.

The static RCMPSP methods are not directly applicable to the dynamic problem since they do not consider the random arrivals of new projects. Two main approaches are available for the dynamic RCMPSP; (1) reactive baseline scheduling (e.g. Pamay et al. (2014)), an approach which generates a baseline schedule

and updates it at each project arrival which allows usage of the static RCMPSP methods such as genetic algorithm for the dynamic RCMPSP and (2) computation of optimal policies using approximate dynamic programming (e.g. Parizi et al. (2017)). Chapter 4 applied both methods to the dynamic RCMPSP and evaluated their performances. More details about Pamay et al. (2014) and Parizi et al. (2017) is given below.

Pamay et al. (2014) treated the dynamic RCMPSP as an static problem by rescheduling a baseline schedule at each project arrival. They assigned earliness and tardiness costs to each task rather than the completion of projects. They used a local search heuristic method intending to find a baseline schedule that minimises the weighted sum of the earliness and tardiness costs of ongoing projects plus the cost associated with the new project's completion time.

Parizi et al. (2017) considered dynamic projects arrivals controlled by arbitrary arrival distribution and modelled the dynamic RCMPSP with deterministic task durations as a discrete-time MDP. The state-space size, which represents the queue capacity, limits the maximum number of unfinished projects in the model. The new projects are rejected after queue capacity is full. They suggested a simulation-based policy iteration method and used a least-squares fitting to tune the parameters of value functions approximation. The goal of the algorithm was to maximise the immediate expected profit. The profit function was the project completion reward minus the rejected project costs, the holding cost for non-active projects and the operation costs for active projects. The state information included the status of tasks whether in a queue, ongoing or completed. They used simulations simulation profits to train their approximate dynamic programming model via linear regression. Then the model is used for online decision making.

2.5 Dynamic and stochastic RCMPSP

The stochastic RCMPSP and the dynamic RCMPSP are created to represent the business environment better than the RCPSP, but they consider different uncertainties of the environment. However, these uncertainties are usually features that exist at the same time for most business such as engineering services, software development, IT services, construction and R&D. By considering the uncertainties of stochastic task duration and dynamic project arrivals together, the environment could be modelled better than the stochastic RCMPSP and the dynamic RCMPSP. The RCMPSP with both random project arrivals and uncertain task durations is called the *dynamic and stochastic* RCMPSP. We address dynamic and stochastic RCMPSP in Chapter 5 and Chapter 6.

Limited research has considered the both dynamic project arrivals and stochastic task durations together e.g.: Adler et al. (1995); Cohen et al. (2005); Choi et al. (2007); Melchiors and Kolisch (2009); Fliedner et al. (2012); Capa and Ulusoy (2015); Melchiors (2015); Melchiors et al. (2018); Chen et al. (2019). The dynamic and stochastic RCMPSP aims to find optimal schedules or scheduling policies that maximise the expected total discounted or time-average project reward minus the costs.

Adler et al. (1995); Cohen et al. (2005); Melchiors and Kolisch (2009) took advantage of the well-developed queueing network approach where interdependent resources process project tasks. This requires consideration of projects of relatively simple structure such as tasks requiring the allocation of: a single unit of a single type of resource. More details about these studies are available below:

Adler et al. (1995) proposed a queuing network approach for the dynamic and stochastic RCMPSP and modelled the product development projects as stochastic processing networks where resources are service stations and tasks are jobs in queues of service stations. They considered Multi-type projects differentiated by probability distributions and processing priorities. Also, they considered resource pooling and extra resource capacities for human resource types. For example, 30% of engineering work can be given to technicians, or extra working hours can be applied to engineers. They conducted simulation studies to investigate the factors, which affect the cycle time, such as the number of resources in resource pools and input control policies for project arrivals.

Cohen et al. (2005) modelled the dynamic and stochastic RCMPSP as a queueing network where projects arrive randomly according to Poisson processes, and tasks durations are exponentially distributed. They proposed a Cross-Entropy based method to minimise the average total stay-time in the system and adopted the constant number of projects in process (CONPIP) methodology to manage project arrivals. They defined two types of queues for unprocessed tasks; the first type of queues is resource queues where tasks are ready to process, the second type is a queue where tasks wait for the completion of their predecessor tasks to enter a resource queue. They determined the best task selection policies for resource queues according to performance function values of multiple simulations results.

Melchiors and Kolisch (2009) also followed a queueing network approach to solve dynamic and stochastic RCMPSP for R&D projects where arrival times and task durations are exponentially distributed. The goal of the algorithm is minimising the expected weighted tardiness. They modelled the resource types as resource servers with three-unit capacity. They defined two types of project where project types determine the task network, task numbers (10,20), task distribution

and project arrival distribution. They used five priority rules and Monte-Carlo simulations on two projects test problems, which were combinations of different projects.

Fliedner et al. (2012); Pamay et al. (2014); Capa and Ulusoy (2015) considered the reactive scheduling method which generates a baseline schedule for current projects then updates it at each time a new project arrival disrupts the schedule. More details about these studies are available below:

Fliedner et al. (2012) modelled the dynamic and stochastic RCMPSP problem as a stochastic RCMPSP by rescheduling a baseline schedule with new project arrivals. They considered multiple project types with different exponentially distributed task durations and networks. However, the number of tasks and resource consumptions are the same for all type of projects. Project arrival rates are determined by Poisson processes which use the resource utilisation values. They offered a genetic algorithm with a resource-based policy to minimise the expected average project completion times.

Capa and Ulusoy (2015) also treat dynamic and stochastic RCMPSP as stochastic RCMPSP by rescheduling it after each project arrival and proposed a three-phase bi-objective genetic algorithm. The first phase of the algorithm is an uncertainty assessment to categorise the newly arrived projects for their predicted resource usage deviation levels. The second phase is a bi-objective genetic algorithm to minimise the total sum of absolute deviations (TSAD) and the overall make-span. The algorithm used single and multi-project scheduling approaches together. The single project approach is used to generate a schedule only for the newly arrived project according to remaining resources. The multi-project approach is used to create a schedule for the newly arrived projects and ongoing projects together. The third phase is rescheduling the project activities according to changes.

Pamay et al. (2014) treats the dynamic RCMPSP as an RCMPSP by rescheduling a baseline schedule at each project arrival. They assigned earliness and tardiness costs to each task rather than the completion of projects. They used a local search heuristic method intending to find a baseline schedule that minimises the weighted sum of the earliness and tardiness costs of ongoing projects plus the cost associated with the new project's completion time.

Melchiors et al. (2018) modelled the problem as a Markov decision process (MDP), using dynamic programming to evaluate optimal policies. The solution approach suffers from the curse of dimensionality and thus can only be used for unrealistically small problems. Melchiors et al. (2018) investigated the integration of order acceptance and process planning on the dynamic and stochastic RCMPSP.

They modelled the problem as a Continues-time Markov decision process and allowed dynamic arrivals of multiple type projects according to stochastic Poisson processes. Each project generated with a single task with exponentially distributed durations. In their case study, there was a piece of equipment as a bottleneck. Thus, only one resource type was defined. They used a dynamic programming model, which uses policy iteration with value iteration to determine the optimal policy. The goal was to maximise the long term average profit per unit time. The profit function was the project completion rewards minus holding costs and execution costs. They compared the algorithm on a case where process planning was first and a case where capacity allocations determined the advance process planning.

Choi et al. (2007); Melchiors (2015) formulate the problem as an MDP and design a scheduling policy via approximate dynamic programming which is similar to our approach in Chapter 6. However, our model is notably more comprehensive and allows for solving larger problems with more complex structure, which are closer to those appearing in practice.

Choi et al. (2007) considered the dynamic and stochastic RCMPSP where new project arrivals and task outcomes (success or failure) are uncertain. They considered applications in the agricultural and pharmaceutical industries. Thus they modelled their problems with serial project networks, single resource type, single resource usage per task and no project due dates. They modelled the problem as a discrete-time MDP model and proposed a Q-Learning based approximate dynamic programming approach to maximise the optimal expected final reward. They followed a similar approach with their previous research Choi et al. (2004) where they took the stochastic task durations, uncertain task outcomes and uncertain costs into account. However, they extend that stochastic RCMPSP model by adding dynamic project arrivals, project cancellation and resource idling decisions. The Q-Learning approach uses stochastic simulations with heuristic policies to approximate the state transition rules and their value functions, then records these to a look-up table. The values in the look-up are used for online decision making.

Melchiors (2015, chapter 7) considered the dynamic and stochastic RCMPSP where new project arrives randomly using Poisson processes, and task durations are exponentially distributed. They considered a semi-open project acceptance system by limiting the maximum project number in it. They suggested an approximate dynamic programming algorithm that approximates the bellman's value function with the goal of minimisation of the average costs. They simulated the open system using random policies and use the simulation result to train the approximate value function via linear regression. They conducted experiments

on small problems with two projects with three tasks with a single unit resource capacity of each resource types, a single unit resource usage per task, same project networks for both projects, rejection, holding and processing costs, but no project due dates.

Chen et al. (2019) divided the dynamic and stochastic RCMPSP into states according to the project's completion conditions and then searched best priority rules for each state. They used 20 priority rules with parallel SGS and four objective function related to project durations. In their model, only one project might arrive at a time, and up to two projects might arrive during the entire scheduling time. The arrival of a new project is random arrival time with a uniform distribution. In their experiments problems with two uniform, two beta and one exponentially distributed task durations are used. Also they considered four resource types. They benchmarked on PSPLIB problems.

2.6 Research gaps

Since most of the literature focuses on the static problem, some research gaps exist in the dynamic variants of RCMPSP. Table 2.1 compares the literature that models the dynamic RCMPSP and dynamic and stochastic RCMPSP as MDP models. There were only two solution approaches available for the dynamic RCMPSP, which are reactive baseline scheduling and computation of optimal policies. We applied both methods to the dynamic RCMPSP in Chapter 4. We used dynamic programming as the computation of optimal policies method and GA and a priority rule heuristic as reactive baseline scheduling methods. First time in the literature, we used dynamic programming for the dynamic RCMPSP, and we evaluated performances of dynamic programming, GA and RBA for the DRCMPS.

In the dynamic and stochastic RCMPSP literature, only Melchiors et al. (2018) considered the dynamic programming method. However, they only considered single task projects. In Chapter 5, we filled the gap in the literature by considering the dynamic programming method on RCMPSP, which have multiple tasks in projects. There are multiple researches in the literature that consider the computation of optimal policies for the dynamic and stochastic RCMPSP. However, none of them considered multiple tasks that require more than one type of resource on activation, tasks that consume different amounts of resource, and large projects with a non-sequential project structure at the same time. This consideration identifies Chapter 6's contribution to the literature.

In Table 2.1, for the project network, serial refers to only serial project networks

Table 2.1: Comparison of literature that modelled dynamic RCMPSP as an MDP

	Stochastic task duration	Project network	Solution method	Resource type	Resource capacity	Problem size
Parizi et al. (2017)	no	any	ADP	multiple	multi-unit	large
Chapter 4	no	serial	DP	single	multi-unit	small
Chapter 5	yes	serial	DP	single	multi-unit	small
Melchiors et al. (2018)	yes	single task	DP	single	multi-unit	small
Choi et al. (2007)	yes	serial	Q-learning	single	multi-unit	small
Melchiors (2015, chapter 7)	yes	any	ADP	multiple	single unit	small
Chapter 6	yes	any	ADP	multiple	multi-unit	large

being considered in the research. Any means that the project networks are not limited to serial project networks. For the resource type, multiple and single refer to the number of resource types used in the research. For the resource capacity, multi-unit and single unit refers to the number of resources available for each resource type. The problem sizes are identified as being small if the problems are considered as being solvable by dynamic programming and large if not.

Chapter 3

Methodology literature

This thesis considers the dynamic and stochastic RCMPSP and the dynamic RCMPSP as an infinite horizon Discrete-Time Markov Decision Process (DT-MDP). These problems are categorised as NP-hard. NP-hard refers that these problems can not be solved or verified in polynomial time.

3.1 The problem setting

In the dynamic and stochastic RCMPSP (Chapter 5 and Chapter 6) and the dynamic RCMPSP (Chapter 4), we consider that J project types which share a pool of K types of renewable *resources* to be processed. The available amounts of each type resources are represented by B_k for $k = 1, ..., K$. In Chapter 4 and Chapter 5, we use only a single type resource.

Projects of the same type j share the same features such as arrival probability Λ_j, number of tasks I_j, project network, resource usages $b^k_{j,i}$, project due date F_j, completion reward r_j and tardiness cost w_j. In Chapter 5 and Chapter 6, projects of the same type share minimal possible task duration $t^{min}_{j,i}$, maximal possible task duration $t^{max}_{j,i}$, task duration distribution $\gamma_{j,i}(\cdot)$ which is conditional on the maximal remaining processing time. In Chapter 4, the durations are not stochastic, which can be achieve by setting $t^{max}_{j,i} = t^{min}_{j,i}$.

Only one project from each type may arrive to the system each unit time during processing of the ongoing projects. The project arrival process requires some MDP terms definitions beforehand. Thus it is described in detail in subsection 3.2.1.

A type j project consists of I_j tasks which are bound to each other with a predecessor-successor relationship. The order of precedence between tasks is also called a *project network*. We consider finish to start precedence relations between tasks. The project network is an important factor for project scheduling since a task requires completion of its predecessor tasks $\mathcal{M}_{j,i}$ for processing. In Chapter 4 and

Chapter 5, tasks are processed serially with a successor-predecessor relationship. In Chapter 6, more general types of finish-to-start successor-predecessor task relationships are considered.

Processing a task i from a project type j may require allocation of $b_{j,i}^k$ amount from each resource types during its processing. The total number of allocated resources cannot be higher than B_k. The allocated resources become reusable after completing the task processing. The un-allocated resources are called *free-resources* $B_k^{\text{free}}(s)$, and allocated resources are added to free resources after their assigned task is completed. Allocated resources are removed from free resources when an action to start processing a task is taken. Free resources are important for decision making and we use free resources to determine the feasible set of actions in pre-decision state s. Task processing is also assumed to be *non-preemptive*; thus, it cannot be paused or cancelled, i.e., once a task has begun processing, it does not leave processing until completed.

A task is completed after it is processed for its processing duration in Chapter 4. Task completion in stochastic environment requires some MDP terms definitions beforehand. Thus it is described in detail in subsection 3.2.1.

When all tasks of a project are processed, the project is completed, and a project *reward* r_j is earned. Projects have a time limit until their due date (F_j) which represents the maximum amount of time which can be spent for project completion to obtain its full reward r_j. If the due date is exceeded, the tardiness cost w_j is applied only once, after the project is completed.

3.2 Markov decision process (MDP)

MDP is a mathematical framework for decision-making problems where outcomes are partly random and partly under the control of a decision-maker (Marinescu, 2018).

We can explain the MDP as a process where a decision-maker uses the current system information relevant to the decision-making process to chose an action; then, with the interaction of the action and some random factor, the system changes and provides a new system information and a reward. Here, the system information is called *state* (s). The time when the decision is taken is called *decision epoch*. Available actions (a) for the state s are called *set of actions* $(\boldsymbol{A}(s))$. The system change is called *transition* and the function which defines this transition is called *transition function* $(P(s'|s,a))$. The period which is during the transition occurs is called *transition time*. As a result of the state, action and transition function, the new information which is called future state (s') and the reward which is called

the immediate profit ($R_{s,a}$) are received. The transition function describes how the system evolves from one state to another as a result of decisions and information (Powell, 2011). $P(s'|s,a)$ is the transition function of the system which evolves from current state s due to effect of the action a to future state s'. The MDP has the memoryless property that refers that the future state depends only on the current state and the action.

An MDP may have a finite or infinite horizon. The static problems (RCPSP, stochastic RCPSP, RCMPSP and stochastic RCMPSP) can be modelled as finite horizon MDP since these problems end when all projects are completed. On the other hand, the dynamic problems (dynamic RCMPSP and dynamic and stochastic RCMPSP) can be modelled as infinite horizon MDP since the system never completes because new project arrival possibility always exists. As the infinite horizon MDPs, average or discounted profit options can be used as the goal function in these problems. In this thesis, we considered dynamic RCMPSP (Chapter 5) and dynamic and stochastic RCMPSP (Chapter 6 and Chapter 7); thus, we modelled these problems as infinite horizon MDPs.

3.2.1 Project scheduling as an MDP

The dynamic and stochastic RCMPSP and the dynamic RCMPSP can be modelled as a DT-MDP by considering that decision epochs occur as fixed intervals or as a continuous time MDP by considering decision epochs occurs when system is changed (e.g., task completion, new project arrival). In this thesis, we considered the former. Both of these models are available for the dynamic and stochastic RCMPSP and the dynamic RCMPSP in the literature. We decided to use the DT-MDP because we assumed a system where project manager makes daily decisions based on the daily updated system information.

In a decision epoch of DT-MDP, a decision-maker (i.e. project manager) chooses an action (i.e. task processing decision) based on the state. Then, by the implementation of the action, the transition time begins. During the transition time, the ongoing tasks are processed for one period, and random events occur. In both of the dynamic and stochastic RCMPSP and the dynamic RCMPSP, project arrival is considered as a random event. New projects arrive randomly during the transition time according to an arrival probability based on their project types (λ_j). In the dynamic and stochastic RCMPSP, tasks completions and project completions (which is a result of task completions) are also random events which occur during the transition time. A task may complete processing according to a conditional probability $\gamma_{j,i}$. We assume $\gamma_{j,i}$ is zero till the task is processed at least its minimal possible task duration ($t_{j,i}^{min}$). Then, the completion probability is a function of

the task's remaining processing time to the task's maximal possible task duration ($t_{j,i}^{max}$).

In the dynamic RCMPSP, tasks and projects completions are deterministic events which occur during the transition time. The transition function is expressed in detail in subsection 4.2.4, subsection 5.2.6 and Section 6.2.3.

The above arrival process is a Bernoulli arrival process, with geometrically distributed inter-arrival times T_j with mean λ_j^{-1} and probability mass function $P(T_j = k) = (1 - \lambda_j)^{k-1}\lambda_j; k = 1, 2, \ldots$. In this discrete-time process, due to the memoryless property of the geometric distribution, the probability of an arrival of a type j project within the current time period is $P(T = 1) = \lambda_j$ regardless of the number of time periods since the last arrival. This is the discrete-time analogue of exponentially distributed inter-arrival times in continuous time models.

At the end of a transition time, an immediate profit is received, which is the reward of the completed project minus any tardiness cost. We consider that the immediate profit belongs to the decision epoch before the transition time. The immediate profit $R_{s,a}$ is expressed in detail in subsection 4.2.5, subsection 5.2.7 and subsection 6.2.4.

At a decision epoch, two types of decision problems exist. These are the acceptance decision of the newly arrived projects and the selection of tasks to begin processing. The decision of acceptance of projects is called *order acceptance* problem, and the decision of tasks to begin processing by considering the limited resources is called RCMPSP. We divide the state to two parts which are *pre-acceptance state* (s^*) and *pre-decision state* (s). The *pre-acceptance state* (s^*) corresponds to system information relevant to the acceptance decision of the newly arrived projects, and the *pre-decision state* (s) corresponds to system information relevant to the selection of tasks to begin processing. The pre-acceptance state is explained in detail in Section 6.2.3, and the pre-decision state is described in detail for the dynamic RCMPSP in subsection 4.2.2 and for the dynamic and stochastic RCMPSP in subsection 5.2.3 and Section 6.2.3.

In this thesis, we focus on task scheduling. Thus we simplify the order acceptance problem by considering a semi-open system where the system allows only one project from each project type. The system is full for a project type if a project from that type is in waiting to be processed or in processing, and the system is empty for a project type if a project from that type is completed or does not exist. Newly arrived projects during a transitional time wait in a queue till the transitional time ends. Then, in the pre-acceptance state, if the system is empty for the newly arrived project type, the system accepts that new project. Otherwise, the system rejects the new arrival.

The action (a) for a pre-decision state represents the information of which tasks will begin processing in the next transitional time. An action is described in detail in subsection 4.2.3, subsection 5.2.4 and Section 6.2.3.

At a decision epoch, we also considered a third state which is the *post-decision state* ($\hat{s}$). A post-decision state represents the system information after the chosen action is implemented. A post-decision state is located at the end of a decision epoch when transition time begins. There is no decision available for a post-decision state. The post-decision state is explained in detail in subsection 5.2.5 and Section 6.2.3.

In summary, we defined three states in each decision epoch and their order is the *pre-acceptance state* (s^*), the *pre-decision state* (s) and the *post-decision state* ($\hat{s}$), respectively. The transitions from s^* to s and from s to $\hat{s}$ are immediate, and there are no time lag at these transitions. Also no reward can be received during these transitions. The set of all pre-decision states is called the state space $\mathcal{S}$. All states consist of the information regarding the remaining task processing times $(x'_{j,i}, x_{j,i}, \hat{x}_{j,i})$ to the late completions and the remaining number of periods until project is due (d_j) for all projects.

3.3 Solution methods used in this thesis

In this thesis, we use six different solution approaches, which are: a dynamic programming algorithm, an approximate dynamic programming algorithm, a worst decision algorithm, an optimal reactive baseline algorithm, a genetic algorithm, and a rule based method. Dynamic programming, approximate dynamic programming and worst decision algorithm are scheduling policies methods. Optimal reactive baseline algorithm, genetic algorithm and rule based method are reactive scheduling methods.

3.3.1 Scheduling policies

Dynamic programming (DP)

Dynamic Programming (DP) is a class of methods which uses the Bellman equation to solve optimal control problems in particular MDPs (Sutton and Barto, 2018). The Bellman equation states that the value of a state is the expected reward of state iteration plus the (discounted) reward of expected next state, and it also states that the expected profit of the best action of a state must be equal to profit of the same state under an optimal policy. (Sutton and Barto, 2018). Using the Bellman equation, an optimality problem can be solved by finding the best action

for all of states. In this way, DP divides the complicated problem into simpler decision problems, and by recursively solving the simpler decision problems it reaches to the optimal solution of the complicated problem.

In this research we used dynamic programming value iteration. Value iteration calculates a sequence of value functions (Tijms, 1994). The value function is the expected long term profit of the MDP model. (3.1), which is from Sutton and Barto (2018), shows how Bellman equation can be written from the definition of the value function.

$$\begin{aligned} V_*(s) &= \max_a \mathbb{E}_{\pi^*}[G|s,a] \\ &= \max_a \mathbb{E}_{\pi^*}[\sum_{t=0}^{\infty} \alpha^t R_{s_{t+1},a_{t+1}}|s,a] \\ &= \max_a \mathbb{E}_{\pi^*}[R_{s,a} + \alpha \sum_{t=0}^{\infty} \alpha^t R_{s_{t+2},a_{t+2}}|s,a] \\ &= \max_a \mathbb{E}[R_{s,a} + \alpha V_*(s')|s,a] \\ &= \max_{a \in \boldsymbol{A}(s)} \sum_{s'} p(s'|s,a)[R_{s,a} + \alpha V_*(s')] \end{aligned} \tag{3.1}$$

Here, the first line is the definition of the value function. In the fist line, G is the expected total discounted profit in a discrete-MDP which can be written as $G = R_{s_t,a_t} + \alpha^1 R_{s_{t+1},a_{t+1}} + \alpha^2 R_{s_{t+2},a_{t+2}} + ... = \sum_{t=0}^{\infty} \alpha^t R_{s_{t+1},a_{t+1}}$. In the second line, immediate profit R_{s_t,a_t} can be separated from $\sum_{t=0}^{\infty} \alpha^t R_{s_{t+1},a_{t+1}}$ as $R_{s_t,a_t} + \alpha \sum_{t=0}^{\infty} \alpha^t R_{s_{t+2},a_{t+2}}$. In the third line, $\sum_{t=0}^{\infty} \alpha^t R_{s_{t+2},a_{t+2}}$ is the expected reward of the future state which can be re-written as $V_*(s')$. The expectation in the forth line, can be rewritten as line five which is the Bellman equation.

The above value iteration algorithm which have discounting can be used in a DP algorithm as Algorithm 2. The Bellman equation can also use time-average reward setting as it shown in Algorithm 1. The time-average reward setting is one of the common settings for DP (Sutton and Barto, 2018).

In Algorithm 1 and Algorithm 2, V represents the value function of a pre-decision state s. $p(s'|s,a)$ is the state transition probability. s' stands for the future pre-decision state of s. $V^{old}(s')$ is the value of s' from next decision epoch. β is pre-specified tolerance number. W_{min} and W_{max} are respectively minimum and maximum value changes between two iterations. Δ is the difference between the minimum and the maximum value changes. α is the discount factor. These processes are repeated until the stopping criteria is met. Algorithm 1 stops when the difference between the maximum and the minimum value changes between two iterations is lower than the minimum value change multiplied by the tolerance number. Algorithm 2 stops when the largest value change of being in a state is

Algorithm 1 Time-average reward value iteration

procedure STATE VALUE ITERATION PROCEDURE
$\beta = 0.000001$ ▷ β is the stopping parameter
For each $\forall s \in \mathcal{S}, V^{old}(s) = 0$ ▷ initial state values
do
for $\forall s \in \mathcal{S}$ **do**
$V(s) = \max_{a \in A(s)} \left[R_{s,a} + \sum_{s' \in \mathcal{S}} p(s'|s,a) V^{old}(s') \right]$ ▷ value function
end for
$W_{max} = \max_{s \in \mathcal{S}} [V(s) - V^{old}(s)]$ ▷ maximum value change
$W_{min} = \min_{s \in \mathcal{S}} [V(s) - V^{old}(s)]$ ▷ minimum value change
$\Delta = W_{max} - W_{min}$ ▷ the difference between max. and min. value changes
Update $\forall s \in \mathcal{S}, V^{old}(s) = V(s)$
while $\Delta > \beta \times W_{min}$
end procedure

Algorithm 2 Discounted value iteration

procedure STATE VALUE ITERATION PROCEDURE
$\beta = 0.001$ ▷ β is the stopping parameter
For each $\forall s \in \mathcal{S}, V^{old}(s) = 0$ ▷ initial state values
do
for $\forall s \in \mathcal{S}$ **do**
$V(s) = \max_{a \in A(s)} \left[R_{s,a} + \sum_{s' \in \mathcal{S}} p(s'|s,a) \alpha V^{old}(s') \right]$ ▷ value function
end for
$W_{max} = \max_{s \in \mathcal{S}} [V(s) - V^{old}(s)]$ ▷ maximum value change
Update $\forall s \in \mathcal{S}, V^{old}(s) = V(s)$
while $W_{max} > \beta(1-\alpha)/(2\alpha)$
end procedure

lower than the discounted tolerance number.

DP suffers from "the curse of dimensionality" which means that the number of states and computational requirements expands exponentially with the number of state variables (Sutton and Barto, 2018). Our investigation about the limitations of DP showed that a state space larger than a five projects with two tasks problem is computationally intractable for our hardware which is a desktop computer with Intel i5-6500T CPU with 2.50 GHZ clock speed and 32 GB of RAM.

We use DP algorithm in our research to find the optimal policy which generates the highest profit of the dynamic RCMPSP and Dynamic and Stochastic RCMPSP. The highest profit helped us to compare other algorithms performance. We used the same value iteration method for Chapter 4, Chapter 5 and Chapter 6. However in the DP algorithm in Chapter 6, our goal function is different, thus we use discounting (α) in the value iteration, and we used a different stopping criteria. The DP algorithm which is used in Chapter 4 and Chapter 5 is shown in Algorithm 1, and which is used in Chapter 6 is shown in Algorithm 2.

Worst decision algorithm (WDA)

Worst decision algorithm (WDA) is another value iteration method and quite similar to Algorithm 1. However, the value function of WDA aims to minimise instead of maximise. The value function of WDA is shown in (3.2).

$$V(s) = \min_{a \in A^*(s)} \left[R_{s,a} + \sum_{s' \in \mathcal{S}} p(s'|s,a) V^{old}(s') \right] \tag{3.2}$$

Here, $A^*(s)$ is the set of all feasible non-anticipating actions which does not include the "do not initiate any task" actions (**0**) unless it is the only possible action in the action set. Due to this action rule, WDA produces non-idle policies.

We use WDA in our comparison to include the minimum reward can be achieved with a non-idle policy.

Approximate dynamic programming (ADP)

Approximate dynamic Programming (ADP) is a modelling strategy to overcome "the curse of dimensionality" problem of DP. Also, ADP can be applied to problems where the problem model or transition function is not known (Powell, 2011). ADP manages these by replacing the true value function with a statistical approximation function (Powell, 2009).

In the approximation function, the approximate profit of an assumed best action takes the place of expected profit of the best action. Also, the state transition

from current state to future state with the (assumed) best action is done by the realization of information (random project arrivals and stochastic task durations in our case) by Monte Carlo simulation. Powell (2011) refers to ADP as "optimizing simulator" since it has an optimization step where a decision is chosen and a simulation step where effects of random changes are simulated.

There are multiple strategies to approximate the true value function: for example, Q-learning, real-time dynamic programming (RTDP), approximate value iteration, approximate value iteration using the post-decision state variable (Powell, 2011). Some ADP methods, such as Q-learning, requires keeping the approximate value estimates of visited states in a table which is called a look-up table. Due to the space requirement of storing the look-up table, this method becomes computationally intractable for bigger size problems.

Some ADP methods use basis functions, which are also referred to as linear models; these models capture the behaviour of state transition function (Powell, 2009). Powell (2009) states that basis functions are popular since they are more straightforward than other ADP methods, but the selection of the effective features require careful modelling.

In this thesis, we employ an ADP approach that estimates the true value function ($V^*(s)$) of Bellman's equation given in (3.3) using the approximating linear model in (3.4), such that $\bar{V}(\hat{s}) \approx V^*(s)$. The linear approximation uses two features: the amount of time a project has received processing by the end of the current period; and the amount of resource consumed by a project during the current period. Evaluation of the approximation only requires knowledge of the state, proposed action and model coefficients. This avoids the need to store the value functions for the entire state space. It also avoids the (potentially) expensive calculation of the expectation required in (3.3). Hence the linear approximation is more efficient in terms of memory and computation. This allows for dynamic decision making in larger problems beyond the scope of classical DP and in problems where rapid decisions are required.

$$V^*(s) = \max_{a \in \boldsymbol{A}(s)} \sum_{s' \in \mathcal{S}} P(s'|s,a)[R_{s,a,s'} + \alpha V^*(s')] \tag{3.3}$$

$$\begin{aligned} \bar{V}(\hat{s}) = \sum_{j}^{J} \Bigg\{ \theta_j^1 \sum_{i}^{I} \left\{ \left\{ t_{j,i}^{max} - \hat{x}_{j,i} + 1 \right\} \mathcal{I}\left\{ \hat{x}_{j,i} > 0 \right\} + t_{j,i}^{max} \mathcal{I}\left\{ \hat{x}_{j,i} = 0 \right\} \right\} + \\ \theta_j^2 \sum_{i}^{I} \sum_{k}^{K} \left\{ b_{j,i}^{k} \mathcal{I}\left\{ \hat{x}_{j,i} > 0 \right\} \right\} \Bigg\} \end{aligned} \tag{3.4}$$

These features were chosen according to our comparisons with different approximation functions. In preliminary experiments, we generated eight different approximation functions which are combinations of features including the amount of time spent processing the projects in the system, the amount of resources being used by the projects, the amount of free (unused) resource, the time until the project due date expires and the total remaining processing times across the projects. According to our experiments the 2-feature linear approximation model presented here showed consistently strong performance in comparison to the alternatives across a range different problem settings.

Coefficients θ_j^1 and θ_j^2 are generated using the least-squares fitting method, which minimises the residual between the linear approximation model results and simulated results. The coefficient generation process is summarised in Algorithm 3.

Algorithm 3 ADP

procedure ADP ALGORITHM

$\forall j \in J, \theta_j^1 = \theta_j^2 = 0$, the initial state $s_1 = \mathbf{0}$. ▷ initial values

for $itr = 1$ to $Iteration$ **do** ▷ for each iteration

$\tilde{V}_{sim} = 0$ ▷ $\tilde{V}_{sim}$ is cumulative simulation profit

for $sim = 1$ to $Simulation$ **do** ▷ for each simulation

$\forall j \in J, D_j^1(sim) = \sum_i^I \left\{ \{t_{j,i}^{max} - x_{j,i} + 1\} \mathcal{I}\{x_{j,i} > 0\} + \{t_{j,i}^{max}\} \mathcal{I}\{x_{j,i} = 0\} \right\}$,

$\forall j \in J, D_j^2(sim) = \sum_i^I \left\{ b_{j,i} \mathcal{I}\{x_{j,i} > 0\} \right\}$ ▷ $x_{j,i} \in s_1$

for $t = 1$ to $Period$ **do** ▷ for each simulation period

find $\boldsymbol{A}(s_t)$ for s_t ▷ $\boldsymbol{A}(s_t)$ is the action set for s_t

find $a = \underset{a \in \boldsymbol{A}(s_t)}{\arg\max} \sum_j^J \left\{ \theta_j^1 \sum_i^I \{\{t_{j,i}^{max} - \hat{x}_{j,i} + 1\} \mathcal{I}\{\hat{x}_{j,i} > 0\} + \{t_{j,i}^{max}\} \mathcal{I}\{\hat{x}_{j,i} = 0\}\} + \right.$

$\left. \hookrightarrow \ \theta_j^2 \sum_i^I \sum_k^K \{b_{j,i}^k \mathcal{I}\{\hat{x}_{j,i} > 0\}\} \right\}$

compute $s_{t+1} = s^M(s_t, a, c_{t+1})$ ▷ state iteration via simulation

$\tilde{V}_{sim} \leftarrow \tilde{V}_{sim} + \alpha^{t-1} R_{s_t, a, s_{t+1}}$

end for

$s_1 \leftarrow s_{Period+1}$ ▷ initial state for the next simulation

end for

$\forall j \in J, \theta\_new_j^1 = \theta\_new_j^2 = 0$ ▷ initial new coefficients

$\underset{\theta\_new_j^1, \theta\_new_j^2}{\arg\min} \sum_{sim=1}^{Simulation} \left\{ \tilde{V}_{sim} - \sum_j^J \left(\theta\_new_j^1 D_j^1(sim) + \theta\_new_j^2 D_j^2(sim) \right) \right\}^2$

$\hookrightarrow$ ▷ The linear approximation of the covariance matrix

$\tau = \tau_{harmonic} / (\tau_{harmonic} + itr - 1)$ ▷ harmonic step size

$\forall j \in J, \theta\_old_j^1 = \theta_j^1$ and $\theta\_old_j^2 = \theta_j^2$

$\forall j \in J, \theta_j^1 = (1 - \tau)\theta\_old_j^1 + \tau \theta\_new_j^1$

$\forall j \in J, \theta_j^2 = (1 - \tau)\theta\_old_j^2 + \tau \theta\_new_j^2$

end for

return $\forall j \in J$, θ_j^1 and θ_j^2

end procedure

Here, we train our approximation function with one thousand iterations. In the first iteration, we assume the initial pre-decision state is an empty state with no existing project, and coefficients are zero. In each iteration, we run one hundred simulations, and from each simulation, we collect features D_j^1 and D_j^2 of the initial pre-decision states and cumulative simulated profit (CSP). After simulations are completed, we estimate coefficients ($\forall j \in 1:J \quad \theta_j^1, \theta_j^2$) by minimizing the sum of the squared deviations between the cumulative discounted profits and the linear approximation model (3.4) using a linear least-squares regression method. We call estimated coefficients as θ_new and existing coefficients as θ_old, and we use them in a dynamic step-size function (3.5) to generate coefficients of the new iteration.

$$\forall j \in J, \forall h \in 1:2, \theta_j^h = (1-\tau)\theta\_old_j^h + \tau\theta\_new_j^h \tag{3.5}$$

We generate the dynamic step-size value τ with the harmonic step-size method (3.6). We set harmonic step-size value as $\tau_{harmonic} = Iteration^{0.5}$. [1]

$$\tau = \tau_{harmonic}/(\tau_{harmonic} + itr - 1) \tag{3.6}$$

We use the terminal pre-decision state of the iteration as the initial pre-decision state in the new iteration. After a specific amount of iterations, the final coefficients are used in the linear approximation model for online decision making.

During a simulation, we find the action set $\boldsymbol{A}(s)$ of the current pre-decision state s_t and, select the most profitable action using the objective function in (3.7). If multiple actions have the highest profits, we randomly select among them.

$$\begin{aligned} a = \underset{a \in \boldsymbol{A}(s_t)}{\arg\max} \sum_j^J \Big\{ \theta_j^1 \sum_i^I \{\{t_{j,i}^{max} - \hat{x}_{j,i} + 1\}\mathcal{I}\{\hat{x}_{j,i} > 0\} + \{t_{j,i}^{max}\}\mathcal{I}\{\hat{x}_{j,i} = 0\}\} + \\ \theta_j^2 \sum_i^I \sum_k^K \{b_{j,i}^k \mathcal{I}\{\hat{x}_{j,i} > 0\}\} \Big\} \end{aligned} \tag{3.7}$$

The post-decision state $\hat{s}$ begins with the implementation of the best action. Then random events occurring over the transition time, c_{t+1}, are simulated according to the transition function and the new pre-decision state s' is achieved. If any project completes during the transition time, their profit is added to the cumulative profit with discounting using the discounting function α^{t-1}. This simulation process repeats for a specific amount of periods, and the final pre-decision state is used as an initial pre-decision state in the next simulation.

[1] We chose the step-size value according to Powell (2011). We also tried KESTEN's Stepsize Rule but we received better coefficient values with the harmonic step-size in our tests.

The approximation function evaluated by Algorithm 3 can be used for online decision making for any pre-decision state. First, all actions for the pre-decision state are generated, then expected profits are calculated using the approximation function for each action. The action with the highest profit is used in that pre-decision state. If multiple actions have the highest profits, one of them is selected randomly.

We use ADP in our research since it does not suffer from the curse of dimensionality in contrast to DP, but it considers both the uncertainties of new project arrivals and task durations. Also, ADP allows us to investigate a wider range of project networks, to consider multiple resource types and much bigger size problems.

3.3.2 Reactive scheduling

A reactive scheduling method generates decisions within a deterministic approach without considering the future uncertainties (Pamay et al., 2014). Then, it iteratively fixes its first schedule according to random changes and makes the schedules feasible again (Rostami et al., 2018). The reactive scheduling method converts each state of the dynamic problem to a static problem, and solution methods generate a baseline schedule for each state.

Even though the RCMPSP literature is vast, the majority of the literature focuses on static problems. However, using the reactive scheduling solution methods for the static environment can be applied to dynamic problems. Thanks to the reactive scheduling method, we used ORBA and GA for dynamic and stochastic RCMPSP, and GA for dynamic RCMPSP.

Optimal reactive baseline algorithm (ORBA)

Optimal reactive baseline algorithm (ORBA) is an exact and brute force algorithm for the static RCMPSP and does not consider new project arrivals. ORBA generates every feasible task scheduling order (TSO) of any waiting for processing tasks. A TSO represents the scheduling order of waiting for processing tasks. Then ORBA calculates the reward and makespan of each TSO by simulation. The simulation generates start and finish times of tasks in a TSO processing using the parallel schedule generation scheme (SGS) without considering new project arrivals. Using parallel SGS on a TSO is explained with more detail in Section 3.3.2. Finally, ORBA creates a non-idling action for the current pre-decision state from the TSO with the highest profit. In case of more than one schedule with the highest profit, the algorithm prioritizes the shortest total makespan between these schedules. If the

tie continues, the algorithm randomly selects one schedule. A non-idling action always allocates resources to tasks when it is possible to do so. This process is shown in Algorithm 4.

Algorithm 4 ORBA

procedure
Y_t is the set of tasks $(j,i) \in$ state s_t for which $x_{j,i} = -1$.
ϵ_{Y_t} is the set of all feasible permutations of the set Y_t.
A feasible TSO is a permutation σ of Y_t such that, for any $m < n$ with $\sigma(m) = (j,i)$ and $\sigma(n) = (j,k)$, $k \notin \mathcal{M}_{j,i}$.
$V_{max} = 0$, $makespan_{min} = \inf$, $TSO^* = \emptyset$ ▷ initial values
for $h = 1$ to $|Y_t|!$ **do** ▷ for all permutations of Y_t
if $\epsilon_{Y_t}(h)$ is a feasible TSO **then**
evaluate $\tilde{V}_{sim}(\epsilon_{Y_t}(h))$ and $makespan_{sim}(\epsilon_{Y_t}(h))$
if $(\tilde{V}_{sim}(\epsilon_{Y_t}(h)) \geq V_{max})$ **or** $(\tilde{V}_{sim}(\epsilon_{Y_t}(h)) = V_{max}$ **and** $makespan_{sim}(\epsilon_{Y_t}(h)) < makespan_{min})$ **then**
$TSO^* = \epsilon_{Y_t}(h)$
$V_{max} = \tilde{V}_{sim}(\epsilon_{Y_t}(h))$
$makespan_{min} = makespan_{sim}(\epsilon_{Y_t}(h))$
end if
end if
end for
return TSO^*
end procedure

Here, Y_t is the set of all waiting for processing tasks ($x_{j,i} = -1$) for a given pre-decision state s_t. ϵ_{Y_t} is the set of all feasible permutations of the set Y_t. $\tilde{V}_{sim}(*)$ is the cumulative profit of TSO. $makespan_{sim}(*)$ is the makespan of TSO. TSO^* is the best TSO found so far. V_{max} is the highest profit found among the feasible TSOs. $makespan_{min}$ is the shortest makespan found among the feasible TSOs.

The generated TSO is converted to non-idling action a for given pre-decision state s_t using a serial scheduling generation scheme (SGS). In a SGS, if there are enough free resources to process the first task in the TSO, its action becomes one ($a_{j,i} = 1$) and its resource usage requirements are subtracted from the free resources. The process then repeats for the remaining tasks in order of the TSO. The TSO for the remaining tasks can be used to create future actions for the following periods as long as no new project arrives. If a new project arrival disturbs the system, the current TSO becomes invalid, and ORBA generates a new TSO.

We use ORBA in our comparison since it generates the best possible result of a reactive scheduling method. In addition, comparisons with ORBA also give us insights into the performances of a static environment method in the dynamic environment. ORBA runs in factorial time. Due to the huge computation time

requirement of brute force algorithms, only small size dynamic and stochastic RCMPSP problems can be solved.

Genetic algorithm (GA)

Genetic algorithm (GA) which, was inspired by Charles Darwin's theory of evolution in the 19th century, was developed by Holland (1992). The GA is one of the search algorithms which searches for the global optimum on the solution space by improving the search samples at each iteration (Mori and Tseng, 1997). The GA uses bio-inspired operators (e.g. Elitist selection, Crossover and Mutation) to develop the population, which is a solution set, in each iteration. The GA is the most-used algorithm for project scheduling problems. However, the algorithm is not suitable for dynamic problems, and a reactive scheduling method is required to apply GA to a dynamic problem.

In this thesis, for a given pre-decision state, GA generates one hundred TSOs, which are random permutations of the waiting for processing tasks. The algorithm evaluates simulates each TSO using Parallel TSO, founds profits and makespans of TSOs and ranks the TSOs using these values. Then using the bio-inspired operators, GA iterate the set of TSOs for one hundred times to create better TSOs that have higher profits and lower makespans.

At each iteration, GA creates one hundred new TSOs using elitist selection, crossover and mutation operators. The elitist selection operator copies the top ten percent highest ranked TSOs from the previous generation of TSOs to the new generation of TSOs. Crossover and mutation operators fill the rest of the new generation. The crossover operator randomly selects two TSOs from the previous generation and randomly selects a task inside of the first TSO. The scheduling order of tasks from the initial task [2] to the selected task is copied from the first (selected) TSO, to (make) a new TSO. Then, the crossover operator copies the remaining of tasks of the first (selected) TSO to the new TSO (to after the randomly selected task) but changes the order of these tasks according to the order of these tasks in the second (selected) TSO. The new TSO is always a feasible TSO, since it is created according to the order of tasks in both selected feasible TSOs. The new TSO is mutated by the mutation operator with a fifty percent chance or it is added to the new generation. Under the mutation operation a task is selected at random and the location of this task in the TSO is randomly re-assigned. The new location can not be later than the task's previous order and can not be sooner than its latest to be processed predecessor task. Thus the mutation operator also ensures that the newly generated TSO is a feasible TSO. Then the new TSO is added to the

[2]The initial task represents the first (earliest) to be processed task.

new generation. When size of the new generation reaches one hundred, TSOs are ranked the same as in the first generation. GA iterates the generations one hundred times.

Finally, GA creates a non-idling action for the current pre-decision state from the TSO with the highest profit using serial SGS. Ties between TSOs are managed as same as ORBA. The parameters such as elitist selection ratio, mutation probabilities are taken from Satıç (2014).

We use GA in our comparison since it is the most used solution algorithm for RCPSP according to Karam and Lazarova-Molnar (2013). Also Fliedner et al. (2012) and Capa and Ulusoy (2015) proposed reactive scheduling methods based on GA for the dynamic and stochastic RCMPSP. Thus we include GA to show its performance difference against DP and ADP.

Rule based algorithm (RBA)

Priority rule algorithms determine which tasks will be processed from the waiting for processing tasks according to maximum or minimum values of task features (e.g., resource usage, task duration length, start or end time of task, etc.) These methods are very simple to apply, but their downside is that these methods might be suboptimal. Priority rule algorithms can be categorized as single-pass or multi pass-methods. If the algorithm produces only one schedule it is called a single-pass method, and if it produces multiple schedules it is called a multi-pass method (Kolisch and Hartmann, 1999). Ulusoy (2002) summarised the most popular priority rules as: shortest processing time (SPT), minimum slack (MSLK), most total successor (MTS), greatest resource demand (GRD), latest finish time (LFT), latest start time (LST), resource scheduling method (RSM), greatest rank positional weight (GRPW) and worst case slack (WCS).

In this thesis, our rule-based algorithm (RBA) uses a single-pass priority rule called the longest task first rule. The rule-based algorithm (RBA) prioritises the tasks with longer processing times, and if two tasks have the same duration, the smallest numbered project type, e.g., project type one, is prioritised over type two or type three. The algorithm generates a baseline schedule for each decision state using the priority rule and the serial scheduling scheme. Then, the baseline schedule is converted to an action, similar to ORBA and GA. We use RBA in our comparison to show the performance comparison of a simple and robust heuristic algorithm. We selected the longest task first rule amongst others because it is simple and easy to apply. Also, Frenk and Kan (1987) shows that the longest task first rule has very strong properties of asymptotic optimality.

Schedule generation schemes

Schedule generation schemes (SGS) convert TSOs to schedules that show the tasks' starting and ending dates. There are two SGS methods which are serial SGS and parallel SGS.

Serial SGS assign tasks to schedule according to their order in the TSO. For example, the serial SGS assigns the first task in the TSO for processing to the earliest time in the schedule where all predecessor tasks of the first task are completed, and there are enough resources to process the task throughout its processing duration. Then, the resource usage of the task is subtracted from the available resource amounts during the processing time of the task. After that, serial SGS repeats this process for other tasks according to their order in the TSO.

Parallel SGS assign tasks to schedule according to the earliest possible start times of tasks in the TSO. For example, the parallel SGS investigates if the first task in the TSO can start at the first period (time t) of the schedule. In other words, the parallel SGS checks that if enough resource is available and all predecessor tasks are completed at the time t. Suppose investigations show that the first task in the TSO can begin processing at the time t. In that case, the task is assigned for processing in time t and its resource usage is subtracted from the available resource amounts during its processing time. If investigations show otherwise, the parallel SGS repeat this process for the following tasks in the TSO until all TSO tasks are investigated. Next, assigned tasks are removed from the TSO. Then, if some tasks remain in the TSO, parallel SGS repeats this process for time $t + 1$ and increments the time till no tasks remained in the TSO.

Overall, it may be said that the serial SGS generates policies that represent the order in the TSO while the parallel SGS generates non-idle policies. In this thesis, we used the serial SGS method to measure profits and durations of TSOs in ORBA and GA. We used the parallel SGS method to create a non-idle action from the best TSO generated by ORBA, GA and RBA.

Chapter 4

Performance evaluation of scheduling policies for the DRCMPSP

Abstract

In this study, we consider the dynamic resource-constrained multi-project scheduling problem (DRCMPSP) where projects generate rewards at their completion, completions later than a due date cause tardiness costs and new projects arrive randomly during the ongoing project execution which disturbs the existing project scheduling plan. We model this problem as a discrete Markov decision process and explore the computational limitations of solving the problem by dynamic programming. We run and compare four different solution approaches on small size problems. These solution approaches are: a dynamic programming algorithm to determine a policy that maximises the average profit per unit time net of charges for late project completion, a genetic algorithm which generates a schedule to maximise the total reward of ongoing projects and updates the schedule with each new project arrival, a rule-based algorithm which prioritise processing of tasks with the highest processing durations, and a worst decision algorithm to seek a non-idling policy to minimise the average profit per unit time. Average profits per unit time of generated policies of the solution algorithms are evaluated and compared. The performance of the genetic algorithm is the closest to the optimal policies of the dynamic programming algorithm, but its results are notably suboptimal, up to 67.2%. Alternative scheduling algorithms are close to optimal with low project arrival probability but quickly deteriorate their performance as the probability increases.

keywords : Dynamic programming, Resource constraint, Project scheduling and DRCMPSP.

4.1 Introduction

Project management is crucial for many sectors such as engineering services, software development, IT services, construction and R&D, Grey (2007); Capa and

Ulusoy (2015); Wang et al. (2015); Adhau et al. (2012). However it is a very challenging enterprise in that only 40% of projects are completed within time, 46% of projects are completed within their predicted budget and only 36% of projects realise their full benefit Wellingtone PPM (2018). Many uncertain factors may affect project execution such as new project arrivals. In this environment, problem size grows and becomes intractable for an exact solution; thus, approximation algorithms are generally preferred. This study applies an exact solution method and some approximation methods to project scheduling problems under uncertainty and compare their performances.

"A *project* is a unique, transient endeavour, undertaken to achieve planned objectives, which could be defined in terms of outputs, outcomes or benefits." APM (2012). A project consists of a collection of *tasks* that are connected via network relationships. A *reward* is released as the outcome of a project completion. A project is completed when all of its tasks are processed and an amount of *resources* (e.g. manpower, equipment) is spent over time to process these tasks. Completion of the project beyond a pre-determined *due date* or to lower standards than agreed may cause penalties and loss of prestige and goodwill which are collectively called the *tardiness cost*.

Determining a task processing order to achieve project goals such as completion in the minimum time or completion within a specific time is called *project scheduling problem* (PSP). The PSP is a vast research area which aims at optimising of project duration, resource allocation and cost evaluation Ortiz-Pimiento and Diaz-Serna (2018). In this area, the *resource-constrained project scheduling problem* (RCPSP) is one of the most extensively studied research Creemers (2015). The common goal of the RCPSP is minimising the completion time and *genetic algorithm* (GA) is the most used solution algorithm for this deterministic problem in the literature Karam and Lazarova-Molnar (2013). Well-known RCPSP test problems are available at PSPLIB Kolisch and Sprecher (1997), which is an online RCPSP library (`http://www.om-db.wi.tum.de/psplib`).

Companies usually manage multiple projects simultaneously, and the RCPSP with multiple projects is called *resource-constrained multi-project scheduling problem* (RCMPSP) Adhau et al. (2012). The RCMPSP is a generalisation of the RCPSP, which is an NP-hard class optimisation problem; thus, RCMPSP and the other generalisation of RCPSPs are also categorised as NP-hard. Gonçalves et al. (2008). Two RCMPSP solution approaches exist: (1) the first approach combines projects in parallel with a dummy start-task and a dummy end-task, then solves the problem as a giant RCPSP, (2) the second approach maintains the multiple projects separately Browning and Yassine (2010). The general goal of the RCMPSP

is minimising the total (for the first approach) or the average (for the second approach) completion time Browning and Yassine (2010). A RCMPSP library named MPSPLIB is available at "http://www.mpsplib.com/" which contains sets of problems generated by Homberger Homberger (2012).

The RCPSP and RCMPSP are static, where the data of project arrival times and their type are known before the scheduling begins. However, many companies accept new projects during the processing of ongoing projects Herbots et al. (2007). That deviates from the project plan and leads to missed due dates and associated tardiness costs Capa and Ulusoy (2015). So, instead of focusing only on completion times, projects are modelled with completion rewards and the objective becomes to maximise the expected profit which is the difference between expected completion rewards and expected tardiness costs. The RCMPSP with uncertain project arrivals and deterministic task durations is called *dynamic* RCMPSP (DRCMPSP). Two main approaches are available for the DRCMPSP; (1) reactive baseline scheduling (e.g. Pamay et al. (2014)), an approach which generates a baseline schedule and updates it at each project arrival which allows usage of the static RCMPSP methods such as the GA for the DRCMPSP and (2) computation of optimal policies (e.g. Parizi et al. (2017)).

In this chapter, we consider the DRCMPSP with uncertain project arrivals. We model the problem as an infinite-horizon discrete-time *Markov decision process* (MDP) which is defined by five elements: time horizon, decision state space, action set, transition function and profit function. We generated task processing policies for the DRCMPSP using multiple solution methods. First, we used *dynamic programming value iteration* method to maximise the time-average profit. Second, we used GA to maximise the total completion reward and reactively fixed the schedule distribution for each project arrivals. Third, we used a *rule-based algorithm* (RBA) to generate a policy using the longest task first rule. Finally, we used *worst decision algorithm* (WDP) to generate a non-idling policy which aims to minimise the time-average profit.

We contribute to the literature by (i) developing a DRCMPSP model considering multi-task project types, extending the work of Melchiors et al. Melchiors et al. (2018) who only considered single-task projects, (ii) developing an efficient implementation of the value iteration algorithm in Julia programming language to solve our model with up to 4 project types, (iii) comparing the (exactly) optimal policy of value iteration with the above-mentioned benchmark policies to evaluate the performance gap between solution approaches, and (iv) illustrating that even in simple problems with 2 or 3 project types, the suboptimality gap of benchmark policies commonly used in practice (genetic algorithm and longest-task-first rule)

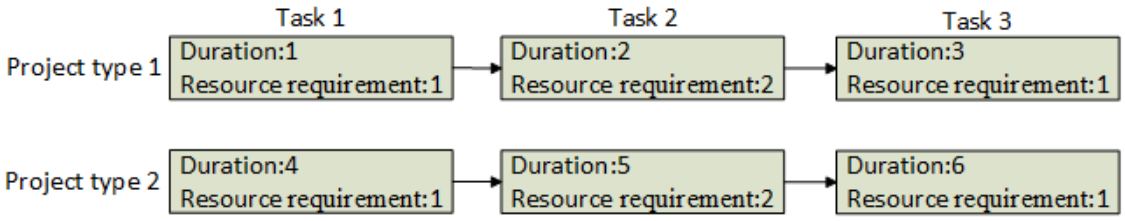

Figure 4.1: A project network

which ignore possibility of new project arrivals is remarkable.

This chapter is organized as follows: In Section 4.2, we describe the problem setting, the MDP model. In Section 4.3, we describe the compared algorithms and discuss comparison results in Section 4.4. In Section 4.5, conclusion is presented.

4.2 Methodology

4.2.1 The problem setting

The DRCMPSP comprises J project types, and the system capacity for each project type is limited to one. All projects of type j share the same characteristics such as arrival probability (λ_j), number of tasks (I_j), task durations ($t_{j,i}$), project network, resource usages ($b_{j,i}$), period till project's due date (F_j), reward (r_j) and tardiness cost (w_j).

A project may arrive to the system at any point during the time unit, which is the duration between two decision epochs. Only one project for each type may arrive per unit time with probability λ_j for a project of type j. Projects are stored in the system until the end of unit time. Then in the next decision epoch, if the system capacity for newly arrived project type is not full, it will get accepted to the system. Otherwise, it will get rejected.

A type j project consists of I_j tasks. In this problem, tasks are connected sequentially with a successor-predecessor relationship, which defines the project network. Processing task i of project type j requires completion of its predecessor tasks ($\mathcal{M}_{j,i}$) which have an earlier place in the project network. An example project network is shown in Figure 4.1.

Processing task i from project type j also requires allocation of $b_{j,i}$ amount of resources during its processing. Only one type of resource is defined in our model and the amount available is represented by B. The total number of allocated resources cannot be higher than B. The resources are assumed renewable which means they become reusable after completion of a task to which they were assigned. The number of resources which are not allocated for task processing is

called free-resources (B_s^{free}). After the completion of a task, its allocated resources return to the free resources.

Task processing is assumed to be non-preemptive; thus, it cannot be paused or cancelled. i.e., once a task has begun processing, it does not leave processing until completed.

Projects are completed when all of their tasks are processed, and a project reward r_j is earned. Projects have a time limit until they are due (F_j) which represents the maximum unit of time which can be spent for project completion to obtain its full reward r_j. If the due date is exceeded, the tardiness cost w_j is applied only once, after the project is completed.

4.2.2 Decision State

The decision state (s) represents the system information relevant to the decision-making process at each decision epoch Sammut and Webb (2010). Decision states where the resource limitations are not exceeded and predecessor tasks were completed before their successor tasks are called feasible and the set of all feasible decision states is called as the state space $\mathcal{S}$. Elements of a decision state $(s = \{\boldsymbol{P}_1, \boldsymbol{P}_2 \ldots, \boldsymbol{P}_J\})$ are project states $(\boldsymbol{P}_j)$ for all project types. A project state consists of task states $(x_{j,i})$ and the due date state (d_j) $(\boldsymbol{P}_j = (x_{j,1}, x_{j,2}, \ldots, x_{j,I_j}, d_j))$.

A task state $(x_{j,i} \in \{-1, 0, 1, 2, \ldots, t_{j,i} - 1\})$ represents the status of a task. If a task is pending for processing, its value is taken as -1. If a task is finished, its value is represented by 0. If a task is in processing, its value is the remaining processing time to its completion.

The due date state $(d_j \in \{0, 1, 2, 3, \ldots, F_j\})$ represents the number of remaining time units from the current time epoch to complete the project j without paying any tardiness cost. When a due date is exceeded, its value becomes 0 and it expresses that the tardiness cost will be incurred at the project's completion. A newly accepted project has the highest due date state value which is F_j.

When a type j project is completed or there is no type j project in the system $(\boldsymbol{P}_j = (0, 0, \ldots, 0, 0))$, all task states $(x_{j,i} = 0, \forall i)$ and due date state $(d_j = 0)$ of project type j are represented by 0.

When a new type j project arrives $(\boldsymbol{P}_j = (-1, -1, \ldots, -1, F_j))$, all its task states are set to -1 $(x_{j,i} = -1, \forall i)$ and its due date state is represented by project's time limit F_j $(d_j = F_j)$.

An example state matrix with two projects and three tasks is shown in Table 4.1. Here, rows of the matrix represent each project type j. The columns represent the task numbers but the last column of the matrix represents the due date state (d_j).

Available resources minus resource usage of a decision state determines free

Table 4.1: State Matrix

		Remaining duration of task i			Remaining duration until project's due date
		1	2	3	d
Project j	1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	d_1
	2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	d_2

resources (B_s^{free}) which is used as a constraint for the available decisions. Free resources are the remaining resources available after the resource allocation to ongoing tasks has been accounted for:

$$B_s^{\text{free}} = B - \sum_{j=1}^{J} \sum_{i=1}^{I_j} b_{j,i} \mathcal{I}\{x_{j,i} > 0\} \tag{4.1}$$

Here, B is the total amount of resource, $b_{j,i}$ is the resource amount allocated for processing of task i from type j project, $\mathcal{I}\{.\}$ is an indicator function that takes the value 1 if the condition in parentheses is true and takes the value 0 otherwise.

4.2.3 Action representation

The decisions available in a given decision state s is called an action a. At a decision epoch, the decision maker selects an action a which starts the processing of the selected pending tasks. An example action matrix with two projects and three tasks is shown in Table 4.2. If the decision includes processing a pending task i of a type j project ($x_{j,i} = -1$), the corresponding action element $a_{j,i}$ will take the value of 1 in the action matrix. Otherwise, $a_{j,i}$ will be 0. The task processing decision can be only taken if there are enough free resources to allocate ($\sum_{j=1}^{J} \sum_{i=1}^{I_j} b_{j,i} \mathcal{I}\{a_{j,i} = 1\} \leq B_s^{\text{free}}$) and any predecessor tasks ($\mathcal{M}_{j,i}$) of task i are completed ($x_{j,m} = 0$ for $\forall m \in \mathcal{M}_{j,i}$). Thus, an action must satisfy both of these conditions.

Table 4.2: Action Matrix

		Task i		
		1	2	3
Project j	1	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
	2	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$

All the actions which meet both the resource and predecessor limitations, are called feasible and set of all feasible actions for a decision state s creates the action set ($\boldsymbol{A}(s) = \{\mathbf{0}, \boldsymbol{a}', \boldsymbol{a}'', \dots\}$).

The action, where all action elements are zero "do not initiate any task" ($\mathbf{0} = (0, 0, \ldots, 0)$) and it is always a member of the action set $\mathbf{0} \in \boldsymbol{A}(s)$. $\boldsymbol{a}^{'}$ and $\boldsymbol{a}^{''}$ represent alternative feasible actions. The number of alternative actions in an action set depends on the number of free resources (B_s^{free}), the unprocessed tasks (with $x_{j,i} = -1$) and the tasks with completed predecessor tasks (with $x_{j,m} = 0$ for $\forall m \in \mathcal{M}_{j,i}$).

4.2.4 Transition function

The transition function describes how the system evolves from one state to another as a result of decisions and information Powell (2011). The period between two consecutive decision states is the time unit. During the transition period; the ongoing tasks are processed for one time unit, some tasks are completed and new projects may arrive according to arrival probabilities λ_j. The project arrival probability is considered when a project is to be completed before the next decision epoch (i.e., the system capacity for type-j project will become available). The transition function is defined in Equation 4.2.

$$P(s'|s,a) = \prod_{j=1}^{J} \prod_{i=1}^{I_j} P(x'_{j,i}|x_{j,i}) \tag{4.2}$$

$$P(x'_{j,i}|x_{j,i}) = \begin{cases} \lambda_j, & \text{for } 0 \leq x_{j,i} \leq 1, x'_{j,i} = -1, i = I_j \\ \lambda_j, & \text{for } x_{j,i} = -1, a_{j,i} = 1, x'_{j,i} = -1, i = I_j \\ 1 - \lambda_j, & \text{for } 0 \leq x_{j,i} \leq 1, x'_{j,i} = 0, i = I_j \\ 1 - \lambda_j, & \text{for } x_{j,i} = -1, a_{j,i} = 1, x'_{j,i} = 0, i = I_j \\ 1, & \text{for } x_{j,i} \geq 2, x'_{j,i} = x_{j,i} - 1, i = I_j \\ 1, & \text{for } x_{j,i} = -1, a_{j,i} = 0, x'_{j,i} = -1, i = I_j \\ 1, & \text{for } x_{j,i} \geq -1, x'_{j,i} \geq -1, i < I_j \end{cases}$$

Here in Equation 4.2, the first two lines represent that, with λ_j probability, there will be an arrival of project type j during the transition time and the new type j project will take the place of the previously completed or non-existing type j project. Due to the sequential project network, completion of the task at the end of a project network represent the project completion. The third and fourth lines represent that, with $1 - \lambda_j$ probability, there will be no new arrival of project type j, which is completed or non-existing, during the transition time. Other lines represent that, with 100% probability, the arrival of projects will not affect the status of ongoing or waiting projects of the same type as the new project will be rejected.

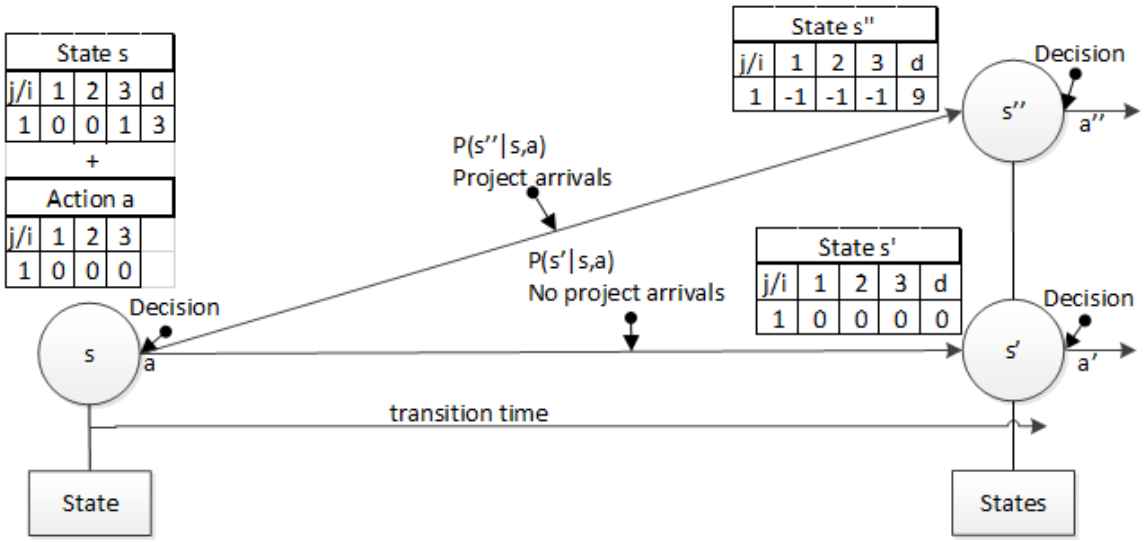

Figure 4.2: A state transition diagram (for a $j = 1$ type project with 3 tasks ($i = 1, 2, 3$) whose due date is $F_j = 9$ and the selected action means do not initialise any task.)

In Figure 4.2 an example transition process has been shown. The transition probability of the first alternative future decision state, where the last task of type j project is finished and a new type j project arrived, is $P(s''|s, a) = \lambda_j$. The transition probability of the second alternative future decision state, where the last task type j project is finished and no project arrived, is $P(s'|s, a) = (1 - \lambda_j)$.

4.2.5 Profit representation

The profit function ($R_{s,a}$) is the sum of rewards (r_j) of completed projects in the period between current and next decision epoch minus the tardiness cost of late completions which depend on the remaining number of periods until project is due.

$$\begin{aligned} R_{s,a} = &\sum_{j=1}^{J} r_j \mathbb{E}\Big[\mathcal{I}\{x_{j,I} = 1 \vee (x_{j,I} = -1 \wedge a_{j,I} = 1 \wedge t_{j,I} = 1)\}\Big] \\ &- \sum_{j=1}^{J} w_j \mathbb{E}\Big[\mathcal{I}\{x_{j,I} = 1 \vee (x_{j,I} = -1 \wedge a_{j,I} = 1 \wedge t_{j,I} = 1) \wedge d_j = 0\}\Big] \end{aligned} \tag{4.3}$$

Here, the first indicator is for project completion and takes the value 1 if a project completes and is 0 otherwise. The project completion occurs when the last task of a project[1] The second indicator is for late project completion. It takes the value

[1]Due to the sequential project network, the task at the end of a project network is the last task of a project. ($x_{j,I}$) is completed. $x_{j,I} = 1$ represents the situation in which the final task of an ongoing project will complete at the end of the current period. $x_{j,I} = -1 \wedge a_{j,I} = 1 \wedge t_{j,I} = 1$ represents the

1 if a project's due date has already passed (i.e., the duration until project's due date $d_j = 0$) and is 0 otherwise. Recall that, in decision state s, $x_{j,I}$ represents the remaining processing time of the final task of a type j project and $a_{j,I}$ its the action element under action a. $t_{j,I}$ is duration of task i of project type j.

4.2.6 Goal function

The goal of the DRCMPSP is to find the policy π that maximises the long-term average profit per unit time. Long-term average is a standard term for Markov decision processing and it refers to $\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T}$. It is also sometimes called long-run average.

$$g^* = \max_{\pi \in \Pi} \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}^{\pi}[R_{s(t),a(t)}] \tag{4.4}$$

Here, t is the time epoch. $R_{s(t),a(t)}$ is the profit function dependent of time epoch t. π is a policy from the set of all feasible non-anticipating policies (Π) presenting the action set $\boldsymbol{A}(s)$. A feasible policy is a sequence of action which considers both the resource limitation and project network.

4.2.7 Solution by dynamic programming

Dynamic Programming is a collection of algorithms which calculates optimal policies from the MDP model of the solution environment Sutton and Barto (2018). In this research we used Dynamic Programming Value Iteration. Value Iteration calculates a sequence of value functions Tijms (1994). The value function approximates the cumulative reward minus the tardiness cost. The per-period change in the value function approximates the maximum long-term average profit. The process steps of the algorithm are below;

For each state $\forall s \in \mathcal{S}, V^{old}(s) = 0$
Do
  For each state $\forall s \in \mathcal{S}$
    $V(s) = \max_{a \in A}[R_{s,a} + \sum_{s' \in \mathcal{S}} p(s'|s,a) V^{old}(s')]$
  End For
  $W_{max} = \max_{s \in \mathcal{S}}[V(s) - V^{old}(s)]$
  $W_{min} = \min_{s \in \mathcal{S}}[V(s) - V^{old}(s)]$
  $\Delta = W_{max} - W_{min}$

situation in which the final task of a project has a single period duration ($t_{j,I} = 1$), is waiting for processing ($x_{j,I} = -1$), and is selected for processing in the current period ($a_{j,I} = 1$).

Update for $\forall s \in \mathcal{S}, V^{old}(s) = V(s)$
While $\Delta > \beta \times W_{min}$

Here, V represents the value function of a decision state s. $R_{s,a}$ is profit function as explained in subsection 4.2.5. $p(s'|s,a)$ is the state transition probability. s' stands for the future decision state of s. $V^{old}(s')$ is the value of s' from next decision epoch. β is pre-specified tolerance number (0.000001). W_{min} and W_{max} are respectively minimum and maximum value changes between two iterations. Δ is the difference between the minimum and the maximum value changes. $\mathcal{S}$ is the state space which is defined at subsection 4.2.2. These processes are repeated until the stopping criteria is met.

4.3 Results and comparisons

We used two heuristic algorithms with reactive scheduling and one worst decision algorithm to compare their performance to optimal. A reactive scheduling method generates decisions within a deterministic approach without considering the future uncertainties Pamay et al. (2014). Then, it iteratively fixes its first schedule according to random changes and makes the schedules feasible again Rostami et al. (2018). We used a genetic algorithm and a priority rule algorithm with the reactive scheduling method.

4.3.1 Genetic algorithm

The discrete-time MDP is considered as a reactive scheduling system by generating a new baseline schedule for each decision state. The baseline schedules are generated by a genetic algorithm (GA) which seeks to maximise the profit and minimising the total completion time. We adapted GA from Satic Satıç (2014). The GA is one of the search algorithms which searches for the global optimum on the solution space by improving the search samples at each iteration Mori and Tseng (1997). The GA uses bio-inspired operators (e.g. Elitist selection, Crossover and Mutation) to develop the population, which is a solution set, in each iteration.

For each decision state, random numbers (between 1 and 30000) are assigned to unprocessed tasks, and this assignment is stored as an individual of the population. Individuals are created until the population number (here, one hundred) is reached. The random numbers represent task processing priorities and this method is called the random key representation. The random keys are converted to a schedule using the serial scheduling scheme as Kolisch and Hartmann Kolisch and Hartmann (1999) described. Then the population is

ordered according to their total profit and total completion time.

The first population is iterated one hundred times using the genetic operators. The best ten percent of the population is transferred to the next population without any change, and the rest of the next population is created with the crossover operator. The crossover operator, firstly, selects two individuals from the previous population, then, copies some random keys from the first individual, after that, copies the rest from another individual, and finally, creates a new individual. The new individual is mutated with a fifty per cent probability before joining to the next population. The mutation operator randomly selects an unprocessed task and re-assigns its random number. When the new population reaches one hundred individuals, the random keys are converted to schedules with the serial scheduling scheme (explained in Section 3.3.2) and the population is ordered again according to their total profit and total completion time. After the one-hundredth generation is created; the best schedule is selected as the baseline schedule. A baseline schedule represents the processing start times of each task. The baseline schedule is converted to an action. The action is the processing decision of tasks, which start at the first time unit on the baseline schedule.

4.3.2 Priority rule (longest task first)

An alternative policy is created with a priority based heuristic algorithm. The algorithm uses a single-pass priority rule called the longest task first rule. Single-pass rules generate only one action for the given state. The rule based algorithm (RBA) prioritises the tasks with the longer processing times and if two tasks have the same duration, the smallest numbered project type, e.g., project type 1 is prioritised over type 2 or type 3. For each decision state, the algorithm generates a baseline schedule using the priority rule and the serial scheduling scheme (explained in Section 3.3.2). Then, the baseline schedule is converted to an action (same as in GA).

4.3.3 Worst decision algorithm

A value iteration method with a non-idling rule is used as the worst decision algorithm (WDP) which seeks a policy (π') to get the minimum profit per unit time. We use WDP in our comparison to include the minimum reward can be achieved with a non-idle policy.

$$g' = \min_{\pi' \in \Pi'} \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}^{\pi'}[R_{s(t),a(t)}]. \tag{4.5}$$

Here, π' is a policy from the set of all feasible non-anticipating active policies (Π') which does not include the "to do not active any task" ($\mathbf{0}$) actions unless it is the only possible action in the action set ($|\boldsymbol{A}(s)| = 1$). Since the reward and tardiness costs are modelled to be received after project completions, a minimum profit algorithm without the priority rule ($|\boldsymbol{A}(s)| \neq 1 \Rightarrow \mathbf{0} \notin \pi'$) delays project completions infinitely to halt rewards.

4.4 Computational results

4.4.1 Experimental setup

Table 4.3: Problem Parameters

2 projects and 2 tasks problem						
Project no	Reward	Tardiness cost	Due date	Task no	Task duration	Resource usage
1	3	1	8	1	2	2
				2	2	2
2	10	9	5	1	3	1
				2	1	3

2 projects and 3 tasks problem						
Project no	Reward	Tardiness cost	Due date	Task no	Task duration	Resource usage
1	12	8	10	1	1	1
				2	2	2
				3	5	1
2	6	5	15	1	4	1
				2	3	2
				3	4	1

3 projects and 2 tasks problem						
Project no	Reward	Tardiness cost	Due date	Task no	Task duration	Resource usage
1	8	5	10	1	5	1
				2	2	1
2	5	3	8	1	1	2
				2	3	1
3	20	19	10	1	2	3
				2	7	2

4 projects and 2 tasks problem						
Project no	Reward	Tardiness cost	Due date	Task no	Task duration	Resource usage
1	18	3	4	1	5	2
				2	1	1
2	27	4	5	1	4	2
				2	2	1
3	18	5	6	1	3	2
				2	3	1
4	18	6	7	1	2	2
				2	4	1

*Resource capacities = 3

In this section, we explore the limits of DP on the DRCMPSP, and compare its

performance with the two heuristic reactive baseline scheduling algorithms and one worst decision algorithm. The DP and the compared algorithms are coded in JuliaPro 1.0.1.1. All tests are performed on a desktop computer with Intel i5-6500T CPU with 2.50 GHZ clock speed and 32 GB of RAM.

We generate four DRCMPSPs (see Table 4.3). For each project in the experiment, a project's tasks are performed in sequential numerical order, i.e., a project starts with task one which is a predecessor of task two which is a predecessor of task three. See Figure 4.1. The problems vary by number of projects, number of tasks, resource usage, different reward-tardiness cost settings and length of due date. We call the difference between a project's due date and the sum of tasks durations as slack time. This value also varies for each project in the problems. The total resource capacity is taken $B = 3$ for all problems.

The first problem has two project types, and each type has two tasks. Project type two has a higher completion reward and higher tardiness cost with a shorter slack time. That means while project type two contributes higher reward opportunities, its late completion is less rewarding compared to the late completion of the project type one.

The second problem has two project types, and each type has three tasks. The project type one is as twice as profitable. However, the slack time of project type one is shorter, so its due date may easily be exceeded leading to tardiness cost.

The third problem has three projects types, and each type has two tasks. In this problem, resource capacity allows parallel processing for only up to two projects increasing the chance of tardiness costs from one project. Only project type one can be processed with other types.

The fourth problem has four projects types, and each type has two tasks. The slack times of project types one and two are negative, and project type three's slack times is zero and project type four's slack time is one. Thus most of the projects will be completed later than their planned due date, and the tardiness payment will be inevitable.

We test each problem consecutively from 1% to 90% project arrival probabilities, increment by 10%. 0% and 100% arrival probabilities are not used in this comparison, because 0% arrival probability makes the problem static and 100% arrival probability causes a non-ergodic MDP, e.g., the empty state where no project has arrived cannot be reachable again from any states. We run each algorithm once on the test problems. However, GA may generate slightly different results in each run. Since our compared problems are very small, GA is able to search most of the state space and the difference between results of different GA runs will be very minor. Thus the results are still competitive.

Table 4.4: Comparison of the time-average profit deviations from the optimal results of DP (how much percent lower than optimal results of DP)

Project arrival probability										
	1%	10%	20%	30%	40%	50%	60%	70%	80%	90%
2 projects and 2 tasks problem										
GA	0.7%	6.5%	11.7%	15.4%	18.1%	20.0%	21.2%	21.4%	20.4%	17.0%
RBA	2.1%	19.9%	35.2%	46.1%	53.7%	59.3%	63.7%	67.3%	70.4%	72.7%
WDP	2.8%	25.6%	43.8%	55.4%	62.7%	67.3%	70.2%	72.1%	73.5%	75.5%
2 projects and 3 tasks problem										
GA	0.1%	4.9%	13.0%	22.0%	31.1%	39.1%	45.6%	51.6%	58.1%	67.2%
RBA	1.5%	15.3%	25.1%	30.1%	32.3%	32.6%	31.1%	28.2%	23.5%	15.4%
WDP	4.1%	34.3%	49.9%	59.0%	66.4%	72.6%	77.1%	80.2%	82.2%	83.3%
3 projects and 2 tasks problem										
GA	0.1%	4.9%	13.0%	22.0%	31.1%	39.1%	45.6%	51.6%	58.1%	67.2%
RBA	1.5%	15.3%	25.1%	30.1%	32.3%	32.6%	31.1%	28.2%	23.5%	15.4%
WDP	4.1%	34.3%	49.9%	59.0%	66.4%	72.6%	77.1%	80.2%	82.2%	83.3%
4 projects and 2 tasks problem										
GA	0.0%	1.2%	2.9%	5.8%	6.9%	6.8%	8.0%	11.5%	15.4%	19.0%
RBA	0.4%	6.6%	14.6%	21.4%	25.1%	26.8%	28.7%	31.4%	33.9%	36.1%
WDP	1.4%	21.3%	37.8%	46.2%	50.5%	52.8%	54.8%	57.3%	59.4%	61.5% [a]

[a] approximate

4.4.2 Discussion

DP suffers from "the curse of dimensionality" which means, here, the number of states grows exponentially with the number of tasks in a project, the number of project types, task durations and due dates, and the large state space becomes computationally intractable Sutton and Barto (2018). The model uses the state space as defined in subsection 4.2.2. In our experiment, a state space for more than five project types with two tasks each becomes computationally intractable. Thus the considered problems are limited to four projects and two tasks.

The results shown in Table 4.4 illustrate that the GA produces almost optimal solutions in 1% arrival rate and produces close to optimal solutions with other low arrival rates. The GA's results are generally closer to optimum compared to RBA for the majority of the considered problems and their task duration variations. The GA's results were from 0.003% to 67.2% lower than the optimum results but never exactly the same.

The RBA's results are between the GA and the WDP for most of the test problem. The RBA's results were from 0.4% to 72.7% lower than the optimum results. In three projects with two tasks problem, the RBA produced better results than the GA at higher arrival probabilities. However, in most of the cases, its

results were closer to the WDP than the optimum since the used priority rule is not designed for reward maximising.

Since the GA and the RBA are reactive baseline scheduling algorithms, they generate decisions without considering the new project arrivals. Thus we may accept that the result of a reactive baseline scheduling algorithm deteriorates compared to the optimum as problem deviates from the static assumption i.e. no project arrivals. However, some anomalies were observed for very high arrival probabilities. These anomalies occur since the tardiness cost is only paid once when a project is completed. In the current model, high arrival probabilities lead to postponing some projects infinitely. Thus, they stay in the system without causing a tardiness cost while the other projects continue processing without causing much tardiness cost.

4.5 Conclusion

In this paper, we studied the resource-constrained multi-project scheduling problem with uncertain project arrivals. We modelled the problem as an infinite-horizon discrete-time MDP. New project arrivals happen during the time unit. We used DP value iteration to maximise the long-term average profit per unit time. We tested the limits of the DP on the DRCMPSP and generated four test problems. We used two heuristic reactive baseline scheduling methods and a worst-decision DP on the same problems and compared their results with exact results of the DP. We used GA and RBA as heuristic reactive baseline scheduling methods.

According to our findings, GA produced closer to optimal results than the simpler heuristic RBA. Since reactive baseline scheduling does not consider the random changes before they occurred, the GA's and the RBA's results are closer to optimal at low arrival probabilities, and diverge from optimum at the high arrival probabilities.

In this work, we have seen that DP suffers from the curse of dimensionality even for the small size problems and reactive baseline scheduling methods do not produce close to optimum results at the high arrival probabilities. Therefore, as a future research topic, we suggest to use a technique which will not (or less) suffer from the curse of dimensionality but will consider the new project arrivals during the decision phase.

Chapter 5

Performance evaluation of scheduling policies for the Dynamic and Stochastic Resource-Constrained Multi-Project Scheduling Problem

Abstract

In this study, we consider the dynamic and stochastic resource-constrained multi-project scheduling problem where projects generate rewards at their completion, completions later than a due date cause tardiness costs, task duration is uncertain, and new projects arrive randomly during the ongoing project execution both of which disturb the existing project scheduling plan. We model this problem as a discrete-time Markov decision process and explore the performance and computational limitations of solving the problem by dynamic programming. We run and compare five different solution approaches, which are: a dynamic programming algorithm to determine a policy that maximises the time-average profit, a genetic algorithm and an optimal reactive baseline algorithm, both generate a schedule to maximise the total profit of ongoing projects, a rule-based algorithm which prioritises processing of tasks with the highest processing durations, and a worst decision algorithm to seek a non-idling policy that minimises the time-average profit. The performance of the optimal reactive baseline algorithm is the closest to the optimal policies of the dynamic programming algorithm, but its results are suboptimal, up to 37.6%. Alternative scheduling algorithms are close to optimal with low project arrival probability but quickly deteriorate their performance as the probability increases.

keywords : dynamic; stochastic; resource constrained project scheduling problem; dynamic programming; reactive scheduling; genetic algorithm; scheduling policies; DSRCMPSP

5.1 Introduction

Many factors may bring uncertainty to the project execution plan, such as new projects arriving after the plan has begun requiring a re-evaluation of the execution order. Project management is a very challenging enterprise in that only 40% of projects are completed within their planned time, 46% of projects are completed within their predicted budget and only 36% of projects realise their full benefit Wellingtone PPM (2018). Many sectors suffer from the uncertainty ignored by traditional project management such as engineering services, software development, IT services, construction and R&D. In this chapter, we propose a comprehensive model to project scheduling under uncertainty for the dynamic and stochastic resource-constrained multi-project scheduling problem (RCMPSP) which includes random project arrival and stochastic task duration uncertainties.

We model the problem as an infinite-horizon discrete-time *Markov decision process* (MDP) with the objective to maximise the expected time-average profit. We extend the research of Chapter 4 by considering stochastic task durations and use their problems in our comparisons by adding early, normal and late task completion probabilities.

The resource-constrained project scheduling problem (RCPSP) and its multi-project equivalent RCMPSP generally consider a static environment. The RCMPSP is a generalisation of the RCPSP, which is an NP-hard optimisation problem; thus, RCMPSP and the other generalisation of RCPSPs are also categorised as NP-hard. (Gonçalves et al., 2008). In the literature, the generalisations of RCPSP and RMCPSP with uncertainty in task durations are called the *stochastic* RCPSP (SRCPSP) and the *stochastic* RCMPSP (SRCMPSP) respectively. The common goal of deterministic problems is minimising the total completion time (Browning and Yassine, 2010), while the common goal of stochastic problems is to minimise the *expected* completion time (Rostami et al., 2018). These static environment problems are extensively studied in the literature. The literature review of Ortiz-Pimiento and Diaz-Serna (2018) shows that Meta-heuristic methods such as GA, particle swarm optimisation, Tabu search, Bee Colony, Ant Colony, Greedy Algorithms, Simulated annealing and Distribution Estimation Algorithm; Exact Methods such as Branch and Bound, Dynamic Programming and Stochastic programming; Special Procedures such as Priority Rules-based, Simulation Process-based or Stage-by-Stage Analysis based; Critical Chain Methods are some of the applied solution methods in the literature. The Critical Chain Method is used for stochastic RCPSP with buffer sizing method as in Zarghami et al. (2019). Also, an approximate dynamic programming method is used for stochastic RCPSP by Li et al. (2020). The literature review of (Karam and Lazarova-Molnar, 2013) on

recent approaches shows that the majority of the new approaches are hybrid forms of previously mentioned methods.

All the models we have described until this point were static, where the data of project arrival times and their type are known before the scheduling begins. Despite the vast number of studies in the static field, many companies accept new projects during the processing of ongoing projects (Herbots et al., 2007). That deviates from the project plan and leads to missed due dates and associated tardiness costs (Capa and Ulusoy, 2015). So, instead of focusing only on completion times, projects are more generally modelled with completion *due dates*, completion *rewards* released as the outcome of project completion, and penalties or loss of prestige and goodwill which are collectively called the *tardiness costs* incurred if projects are completed after their due dates. The aim then becomes to find optimal schedules or scheduling policies that maximise some function of *profit*, which is the difference between the completion rewards and tardiness costs. In general, there are three standard objective functions for non-static problems: (i) the expected total profit over a finite horizon (to the best of our knowledge, this has not been used in projects scheduling literature for non-static problems), (ii) the expected total discounted profit over a finite or infinite horizon (e.g., Parizi, Gocgun, and Ghate 2017), or (iii) the expected time-average profit over an infinite horizon (e.g., Wang et al. 2015). The RCMPSP with uncertain project arrivals and deterministic task durations is called *dynamic* RCMPSP (DRCMPSP). Two main approaches are available for the DRCMPSP; (1) reactive baseline scheduling (e.g. Pamay et al. (2014)), an approach which generates a baseline schedule and updates it at each project arrival which allows usage of the static RCMPSP methods such as the GA for the DRCMPSP and (2) computation of optimal policies using approximate dynamic programming (ADP) (e.g. Parizi et al. (2017)). Chapter 4 applied both methods to DRCMPSP and evaluated their performances.

The RCMPSP with both random project arrivals and uncertain task durations is called the *dynamic and stochastic* RCMPSP. Only a limited number of research considered both the dynamic project arrivals and stochastic task durations together. The main approaches for this problem are processing networks (e.g. Adler et al. (1995); Cohen et al. (2005)), computation of optimal policies using approximate dynamic programming (ADP) (e.g. Melchiors (2015); Choi et al. (2007)) and reactive baseline scheduling (e.g. Fliedner et al. (2012); Capa and Ulusoy (2015)).

We contribute to the literature by (i) developing a dynamic and stochastic RCMPSP model considering multi-task project types, extending the work of Melchiors et al. (2018) who only considered single-task projects, (ii) developing an efficient implementation of the value iteration algorithm in Julia programming

language to solve our model with up to 4 project types, (iii) comparing the (exactly) optimal policy of value iteration with the policies of GA, rule based algorithm and optimal reactive baseline algorithm to evaluate the performance gap between solution approaches, and (iv) illustrating that even in simple problems with 2 or 3 project types, the suboptimality gap of benchmark policies commonly used in practice (genetic algorithm and longest-task-first rule) which ignore possibility of new project arrivals is remarkable.

This chapter is organized as follows: In Section 5.2, we describe the problem setting, the MDP model. In Section 5.3, we describe the compared algorithms and discuss comparison results in subsection 5.4.2. In Section 5.5, a conclusion is presented.

5.2 Methodology

5.2.1 The problem setting

The dynamic and stochastic RCMPSP comprises J project types, and the system capacity for each project type is limited to one. All projects of type j share the same characteristics such as arrival probability (λ_j), number of tasks (I_j), project network, resource usages ($b_{j,i}$), task duration distribution ($\gamma_{j,i}$), minimal possible completion time ($t_{j,i}^{min}$), normal possible completion time ($t_{j,i}$), maximal possible completion time ($t_{j,i}^{max}$), period till project project's due date (F_j), reward (r_j) and tardiness cost (w_j).

A project may arrive to the system at any point during the time unit, which is the duration between two decision epochs. Only one project for each type may arrive per unit time with probability λ_j for a project of type j. Projects are stored in the system until the end of unit time. Then in the next decision epoch, if the system capacity for newly arrived project type is not full, it will get accepted to the system. Otherwise, it will get rejected.

A type j project consists of I_j tasks. In this problem, tasks are connected sequentially with a successor-predecessor relationship, which defines the project network. Processing task i of project type j requires completion of its predecessor tasks ($\mathcal{M}_{j,i}$) which have an earlier place in the project network. An example project network is shown in Figure 5.1.

Processing task i from project type j also requires allocation of $b_{j,i}$ amount of resources during its processing. Only one type of resource is defined in this problem and the amount available is represented by B. The total number of allocated resources cannot be higher than B. The resources are assumed renewable

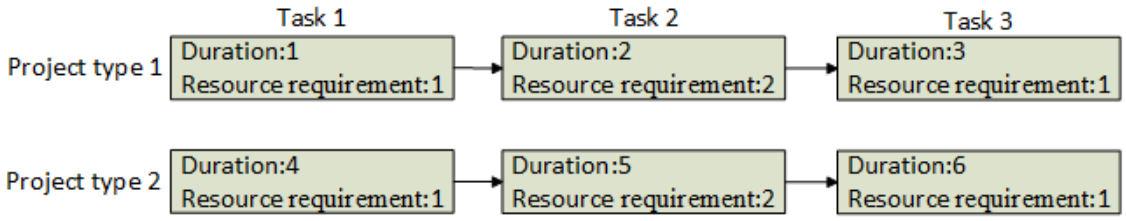

Figure 5.1: A project network

which means they become reusable after completion of a task to which they were assigned. The number of resources which are not allocated for task processing is called free-resources (B_s^{free}). After the completion of a task, its allocated resources return to the free resources.

Task processing is considered as a stochastic process in which tasks can be completed early, normal or late according to its completion distribution ($\gamma_{j,i}$). The distribution $\gamma_{j,i}$ is assumed to be discrete, with the longest (respectively, shortest) processing time with a non-zero probability of completion denoted by $t_{j,i}^{max}$ (respectively, $t_{j,i}^{min}$).

Task processing is also assumed to be non-preemptive; thus, it cannot be paused or cancelled, i.e., once a task has begun processing, it does not leave processing until completed.

Projects are completed when all of their tasks are processed, and a project reward r_j is earned. Projects have a time limit until they are due (F_j) which represents the maximum unit of time which can be spent for project completion to obtain its full reward r_j. If the due date is exceeded, the tardiness cost w_j is applied only once, after the project is completed.

5.2.2 Modelling framework

We model the problem as an infinite horizon *Discrete Time Markov Decision Process* (DT-MDP) which is defined by five elements: time horizon, pre-decision state space, action set, transition function and profit function. Figure 5.2 illustrates this process.

In a DT-MDP, a decision epoch is the time where a decision is taken for a pre-decision state. Decision epochs occur as fixed intervals and the period between two consecutive decision epochs is the time unit. During a time unit, projects are processed according to the decisions made at the previous decision epoch, and the new events occur such as project arrivals and tasks completions may occur. Then the system enters a new decision epoch.

The pre-decision state (s) represents the system information relevant to the

decision-making process at each decision epoch. In this research, the pre-decision state consists of the information regarding the remaining task processing times ($x_{j,i}$) to the late completion and the remaining number of periods until the project is due (d_j) for all projects. More details about the pre-decision state are provided in subsection 5.2.3.

The decisions available in a given pre-decision state s is called action (a). At a decision epoch, the decision maker selects an action a which starts the processing of the selected pending tasks. All actions must fulfil these two conditions; the available free-resources can not be exceeded, and predecessor tasks should be completed. The action is described in detail in subsection 5.2.4.

After the selected action a is applied in the pre-decision state s, system information is represented by the post-decision state $\hat{s} := (s, a)$. The post-decision state is a virtual state before the stochastic processes begin, and it is assumed there is not any time lag between pre-decision state and post-decision state. The post-decision state is explained in subsection 5.2.5.

The transition function describes how the system evolves from one state to another as a result of decisions and information (Powell, 2011). The transition function is illustrated in Figure 5.3 and described in detail in subsection 5.2.6. During the transition period; the ongoing tasks are processed for one time unit, some tasks are completed according to their completion distribution $\gamma_{j,i}$ and new projects may arrive according to arrival probabilities λ_j. Conversely, the probability of no arrival from a project of type j is $1 - \lambda_j$. If a project arrival occurs it will be accepted into the system if there is no project of the same type in the system or, if there is, that active project will complete processing in this time period. Otherwise the new project arrival will be rejected. Hence, between 0 and J project arrivals may occur in a single time unit. We assume that the arrival of projects are independent and identically distributed random variables. The above arrival process is a Bernoulli arrival process, with geometrically distributed inter-arrival times T_j with mean λ_j^{-1} and probability mass function $P(T_j = k) = (1 - \lambda_j)^{k-1}\lambda_j; k = 1, 2, \ldots$.

In this discrete-time process, due to the memoryless property of the geometric distribution, the probability of an arrival of a type j project within the current time period $P(T = 1) = \lambda_j$ regardless of the number of time periods since the last arrival. This is the discrete-time analogue of exponentially-distributed inter-arrival times in continuous time models.

Hence, a stochastic arrival process is employed to model the arrival of new projects. However, for the tractability of solutions via dynamic programming we utilise a finite buffer such that a maximum of one project of each type may be

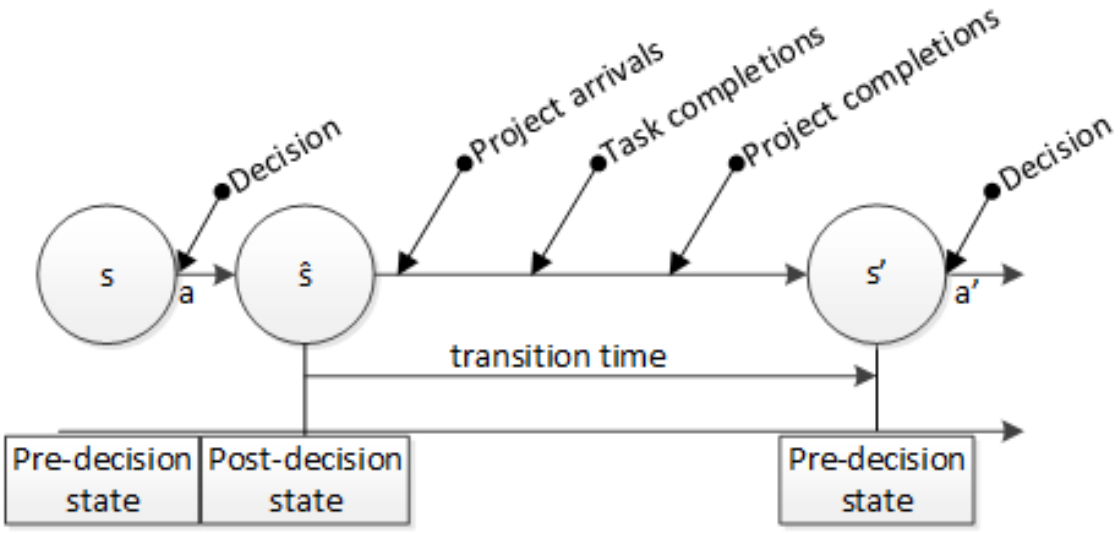

Figure 5.2: Discrete-time Markov Decision Process

present in the system at any point in time.

The profit function of a post-decision state calculates the reward of the completed project minus any tardiness cost paid to its latest possible completion at the end of the transition. The profit calculation is expressed in subsection 5.2.7.

5.2.3 Pre-decision state

The pre-decision state (s) is the system information available at a decision epoch. Pre-decision states where the resource limitations are not exceeded and predecessor tasks were completed before their successor tasks are called feasible and the set of all feasible pre-decision states is called as the state space $\mathcal{S}$. Elements of a pre-decision state are project states ($\boldsymbol{P}_j$) for all project types:

$$s = \{\boldsymbol{P}_1, \boldsymbol{P}_2 \ldots, \boldsymbol{P}_J\} \tag{5.1}$$

A project state consists of task states ($x_{j,i}$) and the due date state (d_j):

$$\boldsymbol{P}_j = (x_{j,1}, x_{j,2}, \ldots, x_{j,I_j}, d_j) \tag{5.2}$$

A task state represents the status of a task. If a task is pending for processing, its value is taken as -1. If a task is finished, its value is represented by 0. If a task is in processing, its value is the remaining processing time to its late completion duration ($t_{j,i}^{max}$). In a pre-decision state, $x_{j,i} = t_{j,i}^{max} - 1$ represents the task processing

Table 5.1: State Matrix

	Task state			Due date state
Project type 1 :	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	d_1
Project type 2 :	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	d_2

began at the previous decision epoch.

$$x_{j,i} \in \{-1, 0, 1, 2, \ldots, t_{j,i}^{max} - 1\} \tag{5.3}$$

The due date state d_j represents the number of remaining time units from the current time epoch to complete the project j without paying any tardiness cost. When a due date is exceeded, its value becomes 0 and it expresses that the tardiness cost will be incurred at the project's completion. A newly accepted project has the highest due date state value which is project's time limit F_j.

$$d_j \in \{0, 1, 2, 3, \ldots, F_j\} \tag{5.4}$$

When a type j project is completed or there is no type j project in the system, all task states ($x_{j,i} = 0, \forall i$) and due date state ($d_j = 0$) of project type j are represented by 0:

$$\boldsymbol{P}_j = (0, 0, \ldots, 0, 0) \tag{5.5}$$

When a new type j project arrives, all its task states are set to -1 ($x_{j,i} = -1, \forall i$) and its due date state is represented by F_j ($d_j = F_j$):

$$\boldsymbol{P}_j = (-1, -1, \ldots, -1, F_j) \tag{5.6}$$

An example state matrix with two project types and three tasks is shown in Table 5.1. Here, rows of the matrix represent each project type j. The columns represent the task numbers but the last column of the matrix represents the due date state (d_j).

A pre-decision state determines its free resources (B_s^{free}) which is used as a constraint for the available decisions. Free resources are the remaining resources available after the resource allocation to ongoing tasks has been accounted for:

$$B_s^{\text{free}} = B - \sum_{j=1}^{J} \sum_{i=1}^{I_j} b_{j,i} \mathcal{I}\{x_{j,i} > 0\} \tag{5.7}$$

Here, B is the total amount of resource, $b_{j,i}$ is the resource amount allocated for processing of task i from type j project, $\mathcal{I}\{.\}$ is an indicator function that takes the

Table 5.2: Action Matrix

	Actions		
Project type 1 :	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
Project type 2 :	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$

value 1 if the condition in parentheses is true and takes the value 0 otherwise.

5.2.4 Action representation

An action a is a function of a pre-decision state s and it holds the processing decisions of pending tasks. An example action matrix with two project types and three tasks is shown in Table 5.2. If the decision includes processing a pending task i of a type j project ($x_{j,i} = -1$), the corresponding action element $a_{j,i}$ will take the value of 1 in the action matrix. Otherwise, $a_{j,i}$ will be 0. The task processing decision can be only taken if there are enough free resources to allocate ($\sum_{j=1}^{J}\sum_{i=1}^{I_j} b_{j,i}\mathcal{I}\{a_{j,i} = 1\} \leq B_s^{\text{free}}$) and any predecessor tasks ($\mathcal{M}_{j,i}$) of task i are completed ($x_{j,m} = 0$ for $\forall m \in \mathcal{M}_{j,i}$). Thus, an action must satisfy both of these conditions.

All the actions which meet both the resource and predecessor limitations, are called feasible and set of all feasible actions for a pre-decision state s creates the action set $\boldsymbol{A}(s)$:

$$\boldsymbol{A}(s) = \{\mathbf{0}, \boldsymbol{a}^{'}, \boldsymbol{a}^{''}, \dots\} \tag{5.8}$$

The action, where all action elements are zero is called "do not initiate any task" ($\mathbf{0} = (0, 0, \dots, 0)$) and it is always a member of the action set $\mathbf{0} \in \boldsymbol{A}(s)$. $\boldsymbol{a}^{'}$ and $\boldsymbol{a}^{''}$ represent alternative feasible actions. The number of alternative actions in an action set depends on the number of free resources (B_s^{free}), the unprocessed tasks (with $x_{j,i} = -1$) and the tasks with completed predecessor tasks (with $x_{j,m} = 0$ for $\forall m \in \mathcal{M}_{j,i}$).

5.2.5 Post-decision state

In our model, the post-decision state ($\hat{s}$) is used to represent the task state after a decision is implemented but before any transition time passed. The transition from a pre-decision state to post-decision is a deterministic process. The post-decision state is used in this study to reduce computational effort and to store the stochastic transition outcomes of a pre-decision state and action pair with their occurrence probabilities.

The same due date state is used for a pre-decision state and its following post-

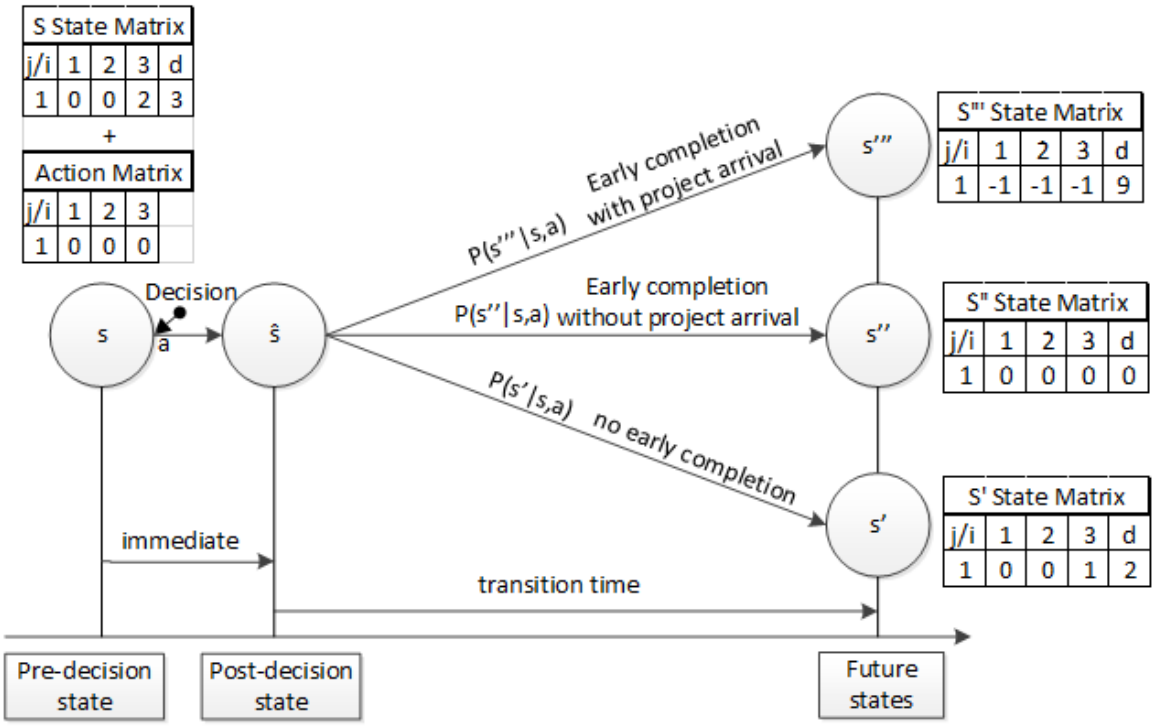

Figure 5.3: A state transition diagram (for a $j = 1$ type project with 3 tasks ($i = 1, 2, 3$) whose project's time limit until its due date is $F_j = 9$ and the selected action means do not initialise any task.)

decision state. Since there is not any time lag between the post-decision state and the previous pre-decision state, the remaining duration of the processing tasks and the state of completed tasks remains the same while the state of the pending tasks selected by the processing decision changes from -1 to $t_{j,i}^{max}$. This represents that these tasks begin processing. Thus the maximum remaining duration of a task in a post-decision state is $t_{j,i}^{max}$ while it is $t_{j,i}^{max} - 1$ in a pre-decision state.

$$\hat{x}_{j,i} \in \{-1, 0, 1, 2, \ldots, t_{j,i}^{max}\} \tag{5.9}$$

5.2.6 Transition function

The transition function determines the following pre-decision state. The task completion probability $\gamma_{j,i}(\hat{x}_{j,i})$ and the project arrival probability λ_j affect the transition probability of the next pre-decision state. In our model a task may complete within $t_{j,i}^{min}$ and $t_{j,i}^{max}$ periods once it has begun processing. These periods represents the task state values $\hat{x}_{j,i} \leq 1 + t_{j,i}^{max} - t_{j,i}^{min}$, since we store the remaining processing times to task's late completion in a task state. The sum of completion probabilities for these task state values is one ($\sum_{x=1}^{1+t_{j,i}^{max}-t_{j,i}^{min}} \gamma_{j,i}(x) = 1$), and the completion probabilities for other task state values are zero ($\gamma_{j,i}(\hat{x}_{j,i}) = 0$ for $\hat{x}_{j,i} > 1 + t_{j,i}^{max} - t_{j,i}^{min}$).

The project arrival probability is considered when it is possible for task I_j to be completed before the next decision epoch, at which point capacity for a newly arriving project of type j will become available. The system transition probability is given by

$$P(s'|s,a) = \prod_{j=1}^{J} \prod_{i=1}^{I_j} P(x'_{j,i}|\hat{x}_{j,i}) \tag{5.10}$$

$$P(x'_{j,i}|\hat{x}_{j,i}) = \begin{cases} \lambda_j \gamma_{j,i}(\hat{x}_{j,i}), & \text{for } 1 \leq \hat{x}_{j,i} \leq 1 + t^{max}_{j,i} - t^{min}_{j,i}, x'_{j,i} = -1, i = I_j \\ (1-\lambda_j)\gamma_{j,i}(\hat{x}_{j,i}), & \text{for } 1 \leq \hat{x}_{j,i} \leq 1 + t^{max}_{j,i} - t^{min}_{j,i}, x'_{j,i} = 0, i = I_j \\ \gamma_{j,i}(\hat{x}_{j,i}), & \text{for } 1 \leq \hat{x}_{j,i} \leq 1 + t^{max}_{j,i} - t^{min}_{j,i}, x'_{j,i} = 0, i < I \\ 1 - \gamma_{j,i}(\hat{x}_{j,i}), & \text{for } 1 \leq \hat{x}_{j,i} \leq 1 + t^{max}_{j,i} - t^{min}_{j,i}, x'_{j,i} = \hat{x}_{j,i} - 1 \\ \lambda_j, & \text{for } \hat{x}_{j,i} = 0, x'_{j,i} = -1, i = I_j \\ 1 - \lambda_j, & \text{for } \hat{x}_{j,i} = 0, x'_{j,i} = 0, i = I_j \\ 1, & \text{for } \hat{x}_{j,i} = 0, x'_{j,i} = 0, i < I_j \\ 1, & \text{for } \hat{x}_{j,i} > 1 + t^{max}_{j,i} - t^{min}_{j,i}, x'_{j,i} = \hat{x}_{j,i} - 1 \\ 1, & \text{for } \hat{x}_{j,i} = -1, x'_{j,i} = -1 \end{cases}$$

In Figure 5.3 an example transition process has been shown. The transition probability of the first alternative future pre-decision state, where the last task of type j project is finished and a new type j project arrived, is $P(s'''|s,a) = \lambda_j \cdot \gamma_{j,i}(\hat{x}_{j,i})$. The transition probability of the second alternative future pre-decision state, where the last task type j project is finished and no project arrived, is $P(s''|s,a) = (1-\lambda_j) \cdot \gamma_{j,i}(\hat{x}_{j,i})$. The transition probability of the third possible alternative future pre-decision state, where the last task type j project is not finished thus a project arrival is not considered, is $P(s'|s,a) = 1 - \gamma_{j,i}(\hat{x}_{j,i})$.

5.2.7 Profit representation

The profit ($R_{\hat{s}}$) of the post-decision state $\hat{s}$ is the sum of rewards (r_j) of completed projects in the period between current and next decision epoch minus the tardiness cost of late completions which depend on the remaining number of periods until project is due.

$$R_{\hat{s}} = \sum_{j=1}^{J} r_j \mathbb{E}\Big[\mathcal{I}\{\hat{x}_{j,I} > 0 \wedge x'_{j,I} \leq 0\}\Big] - \sum_{j=1}^{J} w_j \mathbb{E}\Big[\mathcal{I}\{\hat{x}_{j,I} > 0 \wedge x'_{j,I} \leq 0 \wedge d_j = 0\}\Big] \tag{5.11}$$

Here, the first indicator is for project completion and takes the value 1 if a project completes and is 0 otherwise. The second indicator is for late project completion.

It takes the value 1 if a project's due date has already passed (i.e., the project's remaining dues date $d_j = 0$) and is 0 otherwise. Recall that, in post-decision state $\hat{s}$, $\hat{x}_{j,I}$ represents the remaining maximum processing time of the final task of a type j project. $x'_{j,I}$ is the remaining maximum processing time of the final task of project j at the future pre-decision state (s').

5.2.8 Goal function

The goal of the dynamic and stochastic RCMPSP is to find the policy π that maximises long-term average profit per unit time. Long-term average is a standard term for Markov decision processing and it refers to $\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T}$. It is also sometimes called long-run average.

$$g^* = \max_{\pi \in \Pi} \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}^{\pi}[R_{\hat{s}(t)}] \tag{5.12}$$

Here, $R_{\hat{s}(t)}$ is the profit function dependent of time epoch t. π is a policy from the set of all feasible non-anticipating policies (Π) presenting the action set $\boldsymbol{A}(s)$. A feasible policy is a sequence of action which considers both the resource limitation and project network.

5.2.9 Solution by dynamic programming

Dynamic Programming (DP) is a collection of algorithms which calculates optimal policies from the MDP model of the solution environment (Sutton and Barto, 2018). In this research we used Dynamic Programming Value Iteration. Value Iteration calculates a sequence of value functions (Tijms, 1994). The value function approximates the cumulative reward minus the tardiness cost. The per-period change in the value function approximates the maximum long-term average profit. The process steps of the algorithm are below;

For each state $\forall s \in \mathcal{S}, V^{old}(s) = 0$
Do
  For each state $\forall s \in \mathcal{S}$
    $V(s) = \max_{a \in A}[R_{\hat{s}} + \sum_{s' \in \mathcal{S}} p(s'|s,a) V^{old}(s')]$
  End For
  $W_{max} = \max_{s \in \mathcal{S}}[V(s) - V^{old}(s)]$
  $W_{min} = \min_{s \in \mathcal{S}}[V(s) - V^{old}(s)]$
  $\Delta = W_{max} - W_{min}$
  Update for $\forall s \in \mathcal{S}, V^{old}(s) = V(s)$

While $\Delta > \beta \times W_{min}$

Here, V represents the value function of a pre-decision state s. $R_{\hat{s}}$ is profit function as explained in subsection 5.2.7. $p(s'|s,a)$ is the state transition probability. s' stands for the future pre-decision state of s. $V^{old}(s')$ is the value of s' from next decision epoch. β is pre-specified tolerance number (0.000001). W_{min} and W_{max} are respectively minimum and maximum value changes between two iterations. Δ is the difference between the minimum and the maximum value changes. $\mathcal{S}$ is the state space which is defined at subsection 5.2.3. These processes are repeated until the stopping criteria is met.

5.3 Results and comparisons

We used two heuristic algorithms and one exact algorithm with reactive scheduling and one worst decision algorithm to compare their performance to optimal. A reactive scheduling method generates decisions within a deterministic approach without considering the future uncertainties (Pamay et al., 2014). Then, it iteratively fixes its first schedule according to random changes and makes the schedules feasible again (Rostami et al., 2018). We used a genetic algorithm, an optimal reactive baseline algorithm and a priority rule algorithm; note that all three are based on the reactive scheduling method. Both the optimal DP and the worst decision algorithm are scheduling policies methods.

5.3.1 Genetic algorithm

The GA is one of the search algorithms which searches for the global optimum on the solution space by improving the search samples at each iteration (Mori and Tseng, 1997). The GA uses bio-inspired operators (e.g. Elitist selection, Crossover and Mutation) to develop the population, which is a solution set, in each iteration. The GA is the most-used algorithm for project scheduling problems. However, the algorithm is not suitable for dynamic problems, and a reactive scheduling method is required to apply GA to a dynamic problem. The reactive scheduling method converts each state of the dynamic problem to a static problem, and solution methods generate a baseline schedule for each state. Fliedner et al. (2012) and Capa and Ulusoy (2015) proposed reactive scheduling methods based on GA for the dynamic and stochastic RCMPSP. Thus we included GA to our compared algorithms.

The goal of the genetic algorithm (GA) in this research is maximising the profit. The algorithm uses the total completion time as tiebreakers between schedules

with equal rewards. If the tie continues, the model prioritises processing of lower project type numbers and lower task numbers. We adapted GA from Satıç (2014). For each pre-decision state, random numbers (between 1 and 30000) are assigned to unprocessed tasks, and this assignment is stored as an individual of the population. Individuals are created until the population number (here, one hundred) is reached. The random numbers represent task processing priorities and this method is called the random key representation. The random keys are converted to a schedule using the serial scheduling scheme as Kolisch and Hartmann (1999) described. Then the population is ordered according to their total profit and total completion time. So, the first member of the population represents the best schedule found with highest profit and shortest completion time while the last member represents the worst schedule.

The first population is iterated one hundred times using the genetic operators. The best ten percent of the population is transferred to the next population without any change, and the rest of the next population is created with the crossover operator. The crossover operator, firstly, selects two individuals from the previous population, then, copies some random keys from the first individual, after that, copies the rest from another individual, and finally, creates a new individual. The new individual is mutated with a fifty per cent probability before joining to the next population. The mutation operator randomly selects an unprocessed task and re-assigns its random number. When the new population reaches to one hundred individuals, the random keys are converted to schedules with the serial scheduling scheme (explained in Section 3.3.2). Then the population is ordered from the shortest total completion time to longest, and it is ordered again from the maximum total profit to the minimum. After the one-hundredth generation is created; the first schedule in the population (the best schedule) is selected as the baseline schedule. A baseline schedule represents the processing start times of each task. The baseline schedule is converted to an action. The action is the processing decision of tasks, which start at the first time unit on the baseline schedule.

5.3.2 Optimal reactive baseline algorithm

The optimal reactive baseline algorithm (ORBA) converts each pre-decision state to a static RCMPSP with the reactive scheduling method and generates all possible schedules for the static RCMPSP. The profit and the total makespan of the schedules are calculated using the serial scheduling scheme. The schedule with highest profit is selected as the best schedule and converted to action. In case of more than one schedule with the highest profit, algorithm prioritizes the shortest

total makespan between these schedules. If the tie continues, algorithm randomly selects one schedule. We included the ORBA to show the best possible result of the reactive scheduling method.

5.3.3 Priority rule (longest task first)

An alternative policy is created with a priority based heuristic algorithm. The algorithm uses a single-pass priority rule called the longest task first rule. Single-pass rules generate only one action for the given state. The rule based algorithm (RBA) prioritises the smallest numbered project type, if two tasks have the same duration, e.g., project type 1 is prioritised over type 2 or type 3. For each pre-decision state, the algorithm generates a baseline schedule using the priority rule and the serial scheduling scheme (explained in Section 3.3.2). Then, the baseline schedule is converted to an action (same as in GA). We included the RBA to show the performance of a simple heuristic algorithm.

5.3.4 Worst decision algorithm

A mix of value iteration and priority rule methods are used as the worst decision algorithm (WDP) which seeks a policy (π') to get the minimum profit per unit time (g'). We used this method in our comparison to show the minimum profit of the worst non-idling policy.

$$g' = \min_{\pi' \in \Pi'} \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}^{\pi'}[R_{\hat{s}(t)}]. \tag{5.13}$$

Here, π' is a policy from the set of all feasible non-anticipating active policies (Π') which does not include the "do not active any task" ($\mathbf{0}$) actions unless it is the only possible action in the action set ($|\boldsymbol{A}(s)| = 1$). Since the reward and tardiness costs are modelled to be received after project completions, a minimum profit algorithm without the priority rule ($|\boldsymbol{A}(s)| \neq 1 \Rightarrow \mathbf{0} \notin \pi'$) delays project completions infinitely to halt rewards.

5.4 Computational results

5.4.1 Experimental setup

In this section, we will explore the limits of DP on the dynamic and stochastic RCMPSP, and compare its performance with the two heuristic reactive baseline scheduling algorithms, the optimal reactive baseline algorithm and the worst

decision algorithm. The DP and the compared algorithms are coded in JuliaPro 1.0.1.1. All tests are performed on a desktop computer with Intel i5-6500T CPU with 2.50 GHZ clock speed and 32 GB of RAM.

We will use DRCMPSPs with deterministic task durations from Chapter 4 and generate dynamic and stochastic RCMPSP equivalents with stochastic task duration by adding early and late completion options to these problems. For each project in the experiment, a project's tasks are performed in sequential numerical order, i.e., a project starts with task one which is a predecessor of task two which is a predecessor of task three. See Figure 5.1.

The model uses the state space as defined in subsection 5.2.3. The number of states grows exponentially with the number of tasks in a project, the number of project types, task durations and due dates, and the large state space becomes computationally intractable which is called "the curse of dimensionality" (Sutton and Barto, 2018). In our experiment, a state space for more than five project types with two tasks each becomes computationally intractable. We limited our problem sizes to four project types and two tasks.

The problems considered in our experiments vary by number of project types, number of tasks, resource usages, different reward-tardiness cost settings and lengths of duration until projects' due dates. We call the difference between a duration till project's due date and the sum of its expected tasks durations as slack time. This value also varies for each project in the problems. The total resource capacity is taken $B = 3$ for all problems.

The completion times of the deterministic task duration problem are the normal task completion times ($t_{j,i}$) of the stochastic task duration problem. Further, we assume that a task can complete 1 period earlier ($t_{j,i}^{min} = t_{j,i} - 1$) or later ($t_{j,i}^{max} = t_{j,i} + 1$) than normal in the stochastic version. With completion probabilities uniformly distributed between $[t_{j,i}^{min}, t_{j,i}^{max}]$ when $t_{j,i} \geq 2$; and with $\gamma_{j,i}(2) = 1/3$, $\gamma_{j,i}(1) = 2/3$ when $t_{j,i} = 1$.

We test each problem and its versions consecutively from 1% to 90% project arrival probabilities, increment by 10%. 0% and 100% arrival probabilities are not used in this comparison, because 0% arrival probability makes the problem static and 100% arrival probability causes a non-ergodic MDP, e.g., the empty state where no project has arrived cannot be reachable again from any states. For deterministic task durations and 100% arrival probability, the system can never be empty and results in a cycle. Thus the stopping criteria ($\Delta > \beta \times W_{min}$) of the value iteration (subsection 5.2.9) can no be reachable, the W_{min} value remains as zero, and the W_{max} value does not change after the best policy is found.

Table 5.3: 2 project types and 2 tasks problem.

2 project types and 2 tasks problem						
Project type (j)	Reward (r_j)	Tardiness cost (w_j)	Due date (F_j)	Task no (i)	Normal task duration $(t_{j,i})$	Resource usage $(b_{j,i})$
1	3	1	8	1	2	2
				2	2	2
2	10	9	5	1	3	1
				2	1	3
						Resource capacity = 3

Table 5.4: 2 project types and 2 tasks problem, differences (percent lower) from optimal results of DP.

2 project and 2 task problem with deterministic task durations										
λ_j	1%	10%	20%	30%	40%	50%	60%	70%	80%	90%
ORBA	0.01%	0.5%	1.4%	2.3%	3.1%	4.0%	4.9%	6.0%	7.1%	8.3%
GA	0.7%	6.5%	11.7%	15.4%	18.1%	20.0%	21.2%	21.4%	20.4%	17.0%
RBA	2.1%	19.9%	35.2%	46.1%	53.7%	59.3%	63.7%	67.3%	70.4%	72.7%
WDP	2.8%	25.6%	43.8%	55.4%	62.7%	67.3%	70.2%	72.1%	73.5%	75.5%
2 project and 2 task problem with uniform stochastic task durations (1E 1N 1L)										
λ_j	1%	10%	20%	30%	40%	50%	60%	70%	80%	90%
ORBA	0.7%	4.9%	7.4%	9.0%	10.1%	10.9%	11.5%	11.9%	12.3%	12.5%
GA	1.2%	9.3%	15.0%	18.5%	20.6%	21.8%	22.5%	22.8%	22.9%	22.8%
RBA	2.0%	17.5%	30.1%	39.2%	45.8%	50.6%	54.3%	57.0%	59.0%	60.2%
WDP	2.4%	21.0%	35.1%	44.5%	50.8%	55.1%	58.0%	61.0%	63.6%	65.9%

2 project types and 2 tasks problem

The two project types and two tasks problem (see Table 5.3) is the smallest problem in our test sample with 1424 states. Since project type two has a higher completion reward and higher tardiness cost with a smaller slack time. The project type two contributes larger reward opportunities, however, its late completion is less rewarding compared to the late completion of the project type one. Resource usage of both types of projects allow parallel processing of any task of project type one with the first task of project type two. Thus, the processing decision of the second task of the project type two or any tasks of the project type one, is a bottleneck for this problem.

The minimum difference with optimum is seen at the 1% arrival probability for both version of the problem. The maximum difference is seen at 70% arrival probability for deterministic task durations and 80% arrival probability for stochastic task durations.

Table 5.5: 2 project types and 3 tasks problem.

2 project types and 3 tasks problem						
Project type (j)	Reward (r_j)	Tardiness cost (w_j)	Due date (F_j)	Task no (i)	Normal task duration $(t_{j,i})$	Resource usage $(b_{j,i})$
1	12	8	10	1	1	1
				2	2	2
				3	5	1
2	6	5	15	1	4	1
				2	3	2
				3	4	1
						Resource capacity = 3

Table 5.6: 2 project types and 3 tasks problem, differences (percent lower) from optimal results of DP.

2 project and 3 task problem with deterministic task durations										
λ_j	1%	10%	20%	30%	40%	50%	60%	70%	80%	90%
ORBA	0.003%	0.2%	0.4%	0.6%	0.8%	1.0%	1.1%	1.1%	1.1%	0.8%
GA	0.003%	0.2%	0.4%	0.6%	0.8%	1.0%	1.1%	1.1%	1.1%	0.8%
RBA	0.4%	3.0%	5.0%	7.1%	9.7%	13.2%	17.6%	23.2%	30.6%	40.6%
WDP	0.9%	8.1%	13.6%	18.2%	23.8%	30.0%	36.2%	42.3%	48.2%	53.6%
2 project and 3 task problem with uniform stochastic task durations (1E 1N 1L)										
λ_j	1%	10%	20%	30%	40%	50%	60%	70%	80%	90%
ORBA	0.2%	1.0%	1.5%	1.9%	2.2%	2.6%	2.9%	3.1%	3.3%	3.5%
GA	0.2%	1.0%	1.5%	1.9%	2.2%	2.6%	2.9%	3.1%	3.3%	3.5%
RBA	0.5%	3.0%	4.6%	5.8%	6.9%	7.9%	8.8%	9.6%	10.3%	10.9%
WDP	1.6%	11.5%	17.0%	20.3%	22.6%	24.6%	26.3%	27.7%	28.9%	29.9%

2 project types and 3 tasks problem

The two project types and three tasks problem (see Table 5.5) has 16612 states. Most of the task combination can be processed together, except for second tasks. The project type one is as twice as much profitable. However the slack time of project type one is shorter so its due date may easily be exceeded which leads to pay a tardiness cost.

The GA's results are equal to optimal reactive baseline algorithm's results and both algorithm very close to the optimum at this problem for all arrival probabilities and task duration variations. The RBA's results are less good as its performance deteriorates with higher arrival probabilities.

Table 5.7: 3 project types and 2 tasks problem.

3 project types and 2 tasks problem						
Project type (j)	Reward (r_j)	Tardiness cost (w_j)	Due date (F_j)	Task no (i)	Normal task duration $(t_{j,i})$	Resource usage $(b_{j,i})$
1	8	5	10	1	5	1
				2	2	1
2	5	3	8	1	1	2
				2	3	1
3	20	19	10	1	2	3
				2	7	2
					Resource capacity = 3	

Table 5.8: 3 project types and 2 tasks problem, differences (percent lower) from optimal results of DP.

3 project and 2 task problem with deterministic task durations										
λ_j	1%	10%	20%	30%	40%	50%	60%	70%	80%	90%
ORBA	0.02%	1.7%	6.1%	12.6%	20.0%	26.5%	31.0%	34.1%	36.3%	37.6%
GA	0.1%	4.9%	13.0%	22.0%	31.1%	39.1%	45.6%	51.6%	58.1%	67.2%
RBA	1.5%	15.3%	25.1%	30.1%	32.3%	32.6%	31.1%	28.2%	23.5%	15.4%
WDP	4.1%	34.3%	49.9%	59.0%	66.4%	72.6%	77.1%	80.2%	82.2%	83.3%
3 project and 2 task problem with uniform stochastic task durations (1E 1N 1L)										
λ_j	1%	10%	20%	30%	40%	50%	60%	70%	80%	90%
ORBA	0.3%	4.2%	8.6%	13.2%	18.0%	21.6%	24.4%	26.5%	28.2%	29.5%
GA	0.6%	6.7%	12.8%	19.1%	25.3%	30.3%	33.9%	36.4%	38.0%	38.9%
RBA	1.3%	13.2%	20.4%	23.4%	24.8%	24.8%	24.2%	23.2%	22.1%	21.0%
WDP	3.7%	29.2%	40.6%	45.9%	50.1%	53.3%	55.7%	57.3%	58.3%	59.0%

3 project types and 2 tasks problem

Three project types and two tasks problem (see Table 5.7) has 212568 states. Three types of project can not be processed together and this leads to paying tardiness cost at least for one type of project. A maximum of two project types can be processed at the same time and only project type one can be processed with others. Project type three has the largest reward and highest tardiness cost.

As it can be seen from Table 5.8, the algorithms diverge from the optimum rapidly as arrival probability increases. The GA produced the closest to optimal and to optimal reactive scheduling results at 1% to 30% percent arrival probabilities. After that the RBA generates better results than the GA. Due to high project arrival probabilities, project type one almost always exists in the system, and due to the longest task first rule, RBA always processes project type one. Project type

Table 5.9: 4 project types and 2 tasks problem.

4 project and 2 task problem						
Project type (j)	Reward (r_j)	Tardiness cost (w_j)	Due date (F_j)	Task no (i)	Normal task duration $(t_{j,i})$	Resource usage $(b_{j,i})$
1	18	3	4	1	5	2
				2	1	1
2	27	4	5	1	4	2
				2	2	1
3	18	5	6	1	3	2
				2	3	1
4	18	6	7	1	2	2
				2	4	1
Resource capacity = 3						

two can be processed in parallel with project type one. But the first task of project type three requires all available resources, and it can not start processing while the other projects are processing. Thus the RBA processes type one and type two projects together, and it delays the processing of type three. This allows the RBA to complete project types one and two earning the rewards without paying tardiness costs, while uncompleted project type three never incurs the tardiness cost by the end of the simulation. Thus the RBA achieves a higher profit than the GA.

4 project types and 2 tasks problem

The four project types and two tasks problem (see Table 5.9) is the largest problem in our experiments with 1509132 states. All project types have the same resource usage and sum of task durations. A first task of any project type can be processed with up to two first tasks or one second task of other project types, however, a second task can only be processed with a task one of another project type. The slack times of project one and two are negative, project three's slack times is zero and project four's slack time is one. This implies that most of the projects will be completed later than their planned due date and the tardiness payment will be inevitable.

According to results which is shown in Table 5.10, the GA's results are close to optimal reactive scheduling results. Best results of the alternative algorithms are seen at 1% arrival probability and the worst results are seen at 90% arrival probability.

Table 5.10: 4 project types and 2 tasks problem, differences (percent lower) from optimal results of DP.

4 project and 2 task problem with deterministic task durations										
λ_j	1%	10%	20%	30%	40%	50%	60%	70%	80%	90%
ORBA	0.0003%	0.2%	1.0%	3.2%	4.4%	4.7%	6.4%	10.0%	13.8%	17.8%
GA	0.020%	1.2%	2.9%	5.8%	6.9%	6.8%	8.0%	11.5%	15.4%	19.0%
RBA	0.4%	6.6%	14.6%	21.4%	25.1%	26.8%	28.7%	31.4%	33.9%	36.1%
WDP	1.4%	21.3%	37.8%	46.2%	50.5%	52.8%	54.8%	57.3%	59.4%	61.5% [a]
4 project and 2 task problem with uniform stochastic task durations (1E 1N 1L)										
λ_j	1%	10%	20%	30%	40%	50%	60%	70%	80%	90%
ORBA	0.008%	0.4%	1.5%	3.4%	4.8%	5.7%	6.7%	8.1%	9.4%	10.4%
GA	0.021%	0.8%	2.0%	4.1%	5.7%	6.4%	7.3%	8.5%	9.7%	10.8%
RBA	0.3%	5.3%	11.9%	17.4%	21.0%	23.0%	24.5%	26.0%	27.4%	28.6%
WDP	1.1%	19.1%	34.7%	42.3%	46.3%	48.5%	50.1%	51.5%	52.7%	53.7%

[a] approximate

5.4.2 Discussion

The results shown in Section 5.4 illustrate that none of alternative algorithms produces the optimum results as the DP. However, the ORBA and GA produce almost optimal solutions in 1% arrival probability and close to optimal solutions with the other low arrival probabilities. The ORBA's results were from 0.0003% to 37.6% worse than the optimum results, and they always deteriorate from optimal as the arrival probability increases.

The GA's results are generally closer to optimal reactive scheduling results compared to RBA for the majority of the considered problems and their task duration variations. The GA's results were from 0.003% to 67.2% lower than the optimum results. For the 2 project types and 3 task problem, GA'results are equal to optimal reactive scheduling results. In two of the problems; the results with the deterministic task durations were closer to optimum, while in the other two, it was vice versa.

The RBA's results are between the GA and the WDP for most of the test problem. The RBA's results were from 0.3% to 72.7% lower than the optimum results. In three project types with two tasks problem, the RBA produced better results than the GA at higher arrival probabilities. However, in most of the cases, its results were closer to the WDP than the optimum since the RBA might prioritise the less rewarding project types over others. Thus, it can be said that using a single priority rule usually does not produce good results unless the given priority rule is designed well for problem features.

Since the GA, the ORBA and the RBA are reactive baseline scheduling algorithms, they generate their decisions without considering the future uncertainties

such as early or late task completion or new project arrivals. Thus we may accept that the result of a reactive baseline scheduling algorithm deteriorates compared to the optimum as problem deviates from the static assumption i.e. no project arrivals. However, some anomalies were observed for very high arrival probabilities. These anomalies occur since the tardiness cost is only paid once when a project is completed. In the current model, high arrival probabilities lead to postponing some project types infinitely. Thus, they stay in the system without causing a tardiness cost while the other project types continue processing without causing much tardiness cost.

5.5 Conclusion

Chapter summary. In this chapter, we studied the resource-constrained multi-project scheduling problem with uncertain project arrivals and uncertain task duration. We modelled the problem as an infinite-horizon discrete-time MDP. We inspected both the cases where the task durations are deterministic or stochastic. We used the uniform distribution for the stochastic task durations.

We used DP value iteration to maximise long-term average profit per unit time. We tested the limits of the DP on the dynamic and stochastic RCMPSP and generated four test problems with both deterministic and stochastic task duration variations. Our approach generates the optimum policies for the dynamic and stochastic RCMPSP and contributes to literature by extending the work of Melchiors et al. (2018) which only considered single-task projects. We used two heuristic and one exact reactive baseline scheduling methods and a worst-decision DP on the same problems and compared their results with exact results of the DP. We used GA and RBA as heuristic and ORBA as exact reactive baseline scheduling methods.

According to our findings, a reactive baseline scheduling method with a GA produced closer to optimal results with and without considering arrivals than the priority rule heuristic for the most of the test problem with different arrival probabilities and deterministic or stochastic task durations options. The GA produced the optimal reactive scheduling results for one problem but not for the others even though the setting for GA were the same. The RBA generally produced results between the GA and the WDP. Since reactive baseline scheduling does not consider the random changes before they occurred, the GA's, ORBA's and the RBA's results are closer to optimal at low arrival probabilities and diverge from optimum at high arrival probabilities. The GA's and the RBA's results are closer to optimal at deterministic task durations than the stochastic task durations.

However, a few exceptions have been observed.

Managerial insights. This study provides a performance comparison of the methods and give insights to project managers for determination of the solution method in highly dynamic and stochastic environments. We have seen that DP suffers from the curse of dimensionality even for the small size problems and reactive baseline scheduling methods do not produce close to optimum results at the high arrival probabilities or stochastic task durations. We suggest using DP for small problems and the reactive baseline scheduling methods such as GA for the environments with low uncertainties. We don't recommend using GA or other reactive baseline scheduling methods in highly uncertain environments since our test results showed that GA may generate up to 67.2% less average profit per unit time compared to optimal in these environments. We suggest considering other methods for larger and more complex problems with high or moderate uncertainties.

Future research direction. Our work showed that for environments which change frequently, the most popular method GA and other reactive scheduling methods perform poorly, and alternative solution methods should be considered. Future work might seek other solution methods and compare their performances in these environments. An ADP algorithm with a well-designed and tuned approximation model is a modern example of alternative solution methods. See, for example, Melchiors (2015), Choi et al. (2007), Parizi et al. (2017), which are to the best of our knowledge the only attempts in this direction for similar problems. An extension would be to develop an approximate model and/or approximate solution approach which would not suffer from the curse of dimensionality while also considering both the uncertainties of new project arrivals and task durations as considered here. Other important future research topics are to consider additional uncertainties such as stochastic resource availability or multiple modes of task processing.

Chapter 6

A Simulation-Based Approximate Dynamic Programming Approach to Dynamic and Stochastic Resource Constrained Multi-Project Scheduling Problem

Abstract

We consider the dynamic and stochastic resource constrained multi-project scheduling problem which allows for both random arrival of projects and stochastic task durations. Completion of projects generates rewards, which are reduced by a tardiness cost in case of late completion. Multiple types of resources are available, and projects consume different amounts of these resources when under processing. The problem is modelled as an infinite-horizon discrete-time Markov decision process and seeks to maximise the expected discounted long-run profit. We use an approximate dynamic programming algorithm (ADP) with a linear approximation model which can be used for online decision-making. Our approximation model uses project elements that are easily accessible by a decision-maker with the model coefficients obtained offline via a combination of Monte Carlo simulation and least squares estimation method. Our numerical study shows that ADP and optimal reactive baseline algorithm (ORBA) produce similar results, which are typically both inferior to the optimal results of dynamic programming. ADP has an advantage over ORBA and dynamic programming in that ADP could be applied to larger problems. We also show that ADP generally produces statistically significantly higher profits than common algorithms used in practice, such as a rule-based algorithm and a reactive genetic algorithm.

keywords : Project scheduling; Markov decision processes; approximate dynamic programming; dynamic resource allocation; dynamic programming

6.1 Introduction

Project management and scheduling is challenging. Engineering services, software development, IT services, construction and R&D operate in dynamic environments, often processing multiple projects simultaneously. Many unplanned factors may disturb the project execution plan with new project arrivals and delays in task processing. A project management survey from 2018 shows that only 40% of projects are completed within their planned time, 46% of projects are completed within their predicted budget, and 36% of projects realise their full benefits (Wellingtone PPM, 2018). This chapter considers dynamic arrivals of new projects and stochastic durations of tasks and proposes a comprehensive model and solution approach for the dynamic and stochastic resource-constrained multi-project scheduling problem (*dynamic and stochastic RCMPSP*).

The dynamic and stochastic RCMPSP is a generalisation of the resource-constrained project scheduling problem (RCPSP) which is an NP-hard optimisation problem (Karam and Lazarova-Molnar, 2013). Thus the dynamic and stochastic RCMPSP is also an NP-hard problem. Dynamic refers to random project arrivals from different types of projects and stochastic refers to uncertain task processing times. Dynamic generalisations of RCMPSP are the dynamic RCMPSP and the dynamic and stochastic RCMPSP. A discussion of the RCMPSP and its variants can be found in Chapter 5.

The non-dynamic (i.e., static) variants of RCMPSP are extensively studied (Creemers, 2015). However, the dynamic variants of the RCMPSP where new projects randomly arrive in the system are scarce in the literature. To the best of our knowledge, there are only three research papers available for the dynamic RCMPSP which are Pamay et al. (2014); Chapter 4; Parizi et al. (2017), and there are only ten research papers available for the dynamic and stochastic RCMPSP which are Adler et al. (1995); Cohen et al. (2005); Choi et al. (2007); Melchiors and Kolisch (2009); Fliedner et al. (2012); Capa and Ulusoy (2015); Melchiors (2015); Melchiors et al. (2018); Chen et al. (2019); Chapter 5. They adopt different solution approaches, which have their own strengths and weaknesses.

Adler et al. (1995); Cohen et al. (2005); Melchiors and Kolisch (2009) took advantage of the well-developed queueing network approach where interdependent resources process project tasks. This requires consideration of projects of relatively simple structure such as tasks requiring the allocation of a single unit of a single type of resource. Fliedner et al. (2012); Pamay et al. (2014); Capa and Ulusoy (2015) considered the reactive scheduling method which generates a baseline schedule for current projects and then updates it at each time a new project arrival disrupts the schedule. This approach can be remarkably suboptimal as evidenced in our

computation study in Section 6.5. Melchiors et al. (2018); Chapter 5 modelled the problem as a Markov decision process (MDP), using dynamic programming (DP) to evaluate optimal policies. The solution approach suffers from the curse of dimensionality and thus can only be used for unrealistically small problems. Chen et al. (2019) divided the multi-project problem into states according to the project's completion conditions and then searched best priority rules for each state, but priority rules are notably prone to be suboptimal.

Our methodological approach is similar to Choi et al. (2007); Melchiors (2015); Parizi et al. (2017) in that we also formulate the problem as an MDP and design a scheduling policy via ADP. However, our model is notably more comprehensive and allows for solving larger problems with more complex structure, which are closer to those appearing in practice. Choi et al. (2007) considered applications in the agricultural and pharmaceutical industries; thus, they focused on serial project networks, stochastic task outcomes (success or failure), single resource type, single resource usage per task and no project due dates. Melchiors (2015, chapter 7) conducted experiments on small problems with two projects with three tasks with a single unit resource capacity of each resource types, a single unit resource usage per task, same project networks for both projects, rejection, holding and processing costs, but no project due dates. Parizi et al. (2017) considered deterministic task processing times with rejection, holding and processing costs modelled. Their numerical study had short simulation durations with heavy discounting.

ADP is a powerful tool that provides the researchers with the ability to adjust the complexity of the optimisation model to trade-off the solution complexity of large (realistic) problems at the expense of a modest suboptimality. An acceptable trade-off can be achieved by careful mathematical modelling of the problem in hand; this is in contrast to general purpose methods such as genetic algorithms and other heuristics which typically rely only on tuning of algorithm parameters. Our literature summary shows that ADP has been used in dynamic variants of the RCMPSP such as Choi et al. (2007); Melchiors (2015); Parizi et al. (2017), and static variants of the RCMPSP such as Li and Womer (2015); Li et al. (2020). Outside of applications in project scheduling, ADP methods have been applied areas such as clinical trials (Ahuja and Birge, 2020), vehicle scheduling (He et al., 2018), capacity allocation (Schütz and Kolisch, 2012), machine scheduling (Ronconi and Powell, 2010) and missile defence systems (Davis et al., 2017).

6.1.1 Chapter contributions and outline

We consider that new projects arrive randomly during the ongoing project execution, completion of projects generate rewards, projects arrive with a due date and

completions later than a due date cause tardiness costs, processing times of the project tasks are uncertain, multiple types of resources are available and multiple amounts of resources can exist in each project type. We model the problem as an infinite-horizon discrete-time MDP and seek to maximise the expected discounted long-run profit.

In this chapter we show that ADP is a very useful and advantageous method for the dynamic and stochastic RCMPSP. We use an ADP algorithm with a linear approximation model to approximate the value function of the Bellman equation. Our approximation model uses resource consumption and processing times of project types as features and can be used for online decision making after estimating the coefficients of the linear value function approximation in a simulation-based training phase.

We compare performance of our ADP algorithm with four solution approaches from Chapter 5 a DP algorithm which finds the optimal policy; an optimal reactive baseline algorithm (ORBA) and a genetic algorithm (GA) that both generate schedules to maximise the total profit of ongoing projects; and a rule-based algorithm (RBA) that uses the longest task first rule to guide the allocation of remaining resources to tasks.

We run our benchmark tests on problems of Chapter 5. Also, we generate our comparison problems that are larger and include non-sequential project networks and multiple resource types additional to problems from Chapter 5. The larger size problems are computationally intractable for DP and ORBA; thus, we benchmark ADP with GA and RBA on these problems.

We contribute to the literature by (i) a new comprehensive MDP model which considers the random arrival of new projects, stochastic task durations, multiple resource types, non-sequential project networks, project due dates and tardiness costs, (ii) a new approximation function that uses project rewards, tardiness costs, spent processing time and resource usage of projects for decision making, and it is capable of solving much larger, more complex and much more general problems than ADPs from existing literature, (iii) an extensive simulation study illustrating the strengths and weaknesses of different approaches, (iv) benchmarking with DP and ORBA whenever tractable and with two other approaches in larger problems, (v) developing an efficient implementation of ADP method in Julia programming language to solve dynamic and stochastic RCMPSPs.

This chapter is organized as follows: In Section 6.2, we describe the problem setting, the MDP model and our goal function. In Section 6.3, we describe our ADP algorithm and its coefficient training procedure. In Section 6.4, we describe compared algorithms and discuss comparison results in Section 6.5. In Section 6.6,

the conclusion is presented.

6.2 The Dynamic and stochastic RCMPSP model

6.2.1 Problem setting

We consider K types of renewable *resources* and the available amounts of each type are represented by B_k for $k = 1, ..., K$. *Projects* of the same type j share the same features such as arrival probability Λ_j, number of tasks I_j, project network, resource usages $b^k_{j,i}$, minimal possible task duration $t^{min}_{j,i}$, maximal possible task duration $t^{max}_{j,i}$, task duration distribution $\gamma_{j,i}(\cdot)$ which is conditional on the maximal remaining processing time, project's time limit until its due date F_j, completion reward r_j and tardiness cost w_j.

Projects *arrive* in the system between two decision epochs (transition time) with the arrival probability of their project type Λ_j. The system always accepts newly arrived projects until the system capacity for type j projects are reached and rejects arrivals when capacity is full.

A project is a series of tasks which are bound to each other with a predecessor-successor relationship. The order of precedence between tasks is also called a *project network*. We consider finish to start precedence relations between tasks. The project network is an important factor for project scheduling since a task requires completion of its predecessor tasks $\mathcal{M}_{j,i}$ for processing.

Task processing requires allocation of resources $b^k_{j,i}$. The allocated resources become reusable after completing the task processing. The un-allocated resources are called *free-resources* $B^{\text{free}}_k(s)$, and allocated resources are added to free resources after their assigned task is completed. Allocated resources are removed from free resources when an action to start processing a task is taken. Free resources are important for decision making and we use free resources to determine the feasible set of actions in pre-decision state s. [1]

We only consider *non-preemptive* task processing; thus, task processing cannot be paused or cancelled after it began until its completion. We allow for *stochastic task durations* such that a task may be completed be in some period between t^{min} and t^{max}. The probability that a task completes in the current period is conditional

[1]Since resource availability is deterministic, two methods are available for maintaining free resources. (i) Free resources are not stored, but their values are calculated in pre-decision states as available amounts of resources minus resource usages of ongoing tasks. (ii) Free resources values are stored in the system. Their value is updated by removing allocated resources of an action in post-decision states and adding released resources from completed tasks in pre-acceptance states. In this work, we considered the latter. We compared two methods but did not observe any performance difference between them.

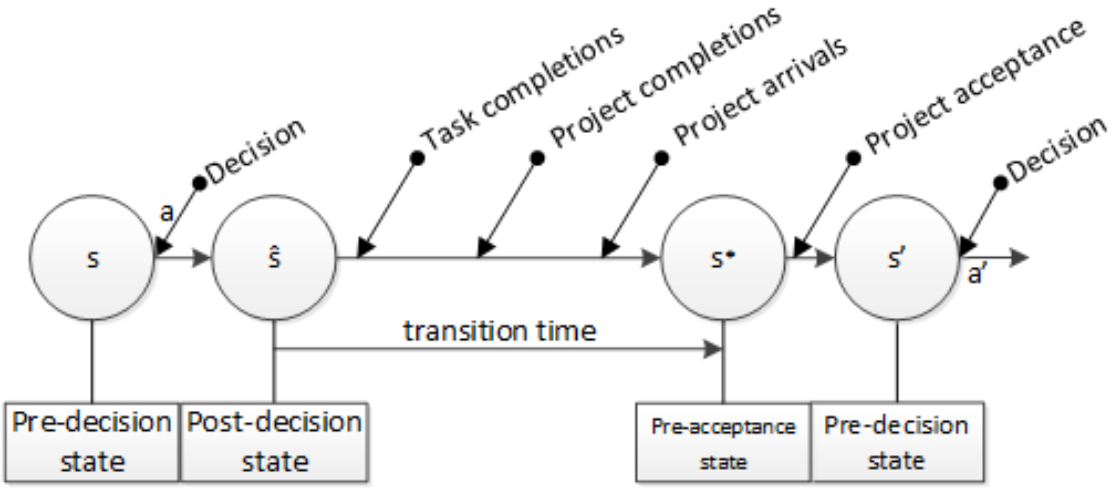

Figure 6.1: Discrete-time Markov Decision Process

on the maximal remaining task processing time $\hat{x}$.

When all tasks of a project are processed, the project is completed, and a project *reward* r_j is earned. However, *tardiness cost* w_j is incurred if the remaining periods before expiration of the project due date d_j is zero at project completion.

6.2.2 Modelling framework

We consider the dynamic and stochastic RCMPSP as an infinite horizon *Discrete Time Markov Decision Process* (DT-MDP) model. A DT-MDP is 5-tuple consisting of state space $\mathcal{S}$, set of actions $\boldsymbol{A}(s)$, transition function $P(s'|s,a)$, the immediate reward $R_{\hat{s}}$ and discount factor α.

The DT-MDP is a discrete decision model where a decision-maker takes action for a state ($s \in \mathcal{S}$) to maximise the discounted profit, and the state randomly changes for a predetermined discrete period after the action is taken. This process repeats for all new states over the infinite horizon. Figure 6.1 illustrates this process.

In this research, we use terms pre-decision state s for a state before the action is taken, and post-decision state $\hat{s}$ for a state after the action is taken and pre-acceptance state s^* for a state immediately before the decision about accepting or rejecting projects that have arrived during the current period. These terms are described in detail below.

We call the moment when an action is taken for a pre-decision state as a *decision epoch*. We call the period between two decision epochs as a *transition time*. We assume the system transfers from a pre-decision to a post-decision state instantly; thus, the transition time is equal to the predetermined discrete period.

6.2.3 Model dynamics

Pre-decision state

For the dynamic and stochastic RCMPSP, a pre-decision state s represents the system information available at a decision epoch. The set of all pre-decision states is called the state space $\mathcal{S}$. A pre-decision state s consists of project states $\boldsymbol{P}_j$ for all project types $j \in J$.

$$s = \{\boldsymbol{P}_1, \boldsymbol{P}_2 \ldots, \boldsymbol{P}_J\} \tag{6.1}$$

A project state consists of task states $x_{j,i}$ of tasks $i \in I_j$ and the due date state d_j. [2]

$$\boldsymbol{P}_j = (x_{j,1}, x_{j,2}, \ldots, x_{j,I_j}, d_j) \tag{6.2}$$

$$s = \begin{bmatrix} x_{1,1}, & x_{1,2}, & \ldots & x_{1,I_1}, & d_1 \\ x_{2,1}, & x_{2,2}, & \ldots & x_{2,I_2}, & d_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{J,1}, & x_{J,2}, & \ldots & x_{J,I_J}, & d_J \end{bmatrix} \tag{6.3}$$

A task state $x_{j,i}$ represents the status of the i^{th} task of project type j. If task i is "pending for processing", its state is -1. If task i is "in processing", its state shows the remaining task processing time to its maximal possible duration $t_{j,i}^{max}$. If task i is "completed", its state becomes 0.

$$x_{j,i} \in \{-1, 0, 1, 2 \ldots, t_{j,i}^{max} - 1\} \tag{6.4}$$

A due date state d_j shows the number of remaining periods from the current decision epoch to the project due date without paying any tardiness cost. In a decision epoch, if a due date state value is zero ($d_j = 0$) while the project still has some uncompleted tasks (i.e., $x_{j,i} = -1$ or $x_{j,i} > 0$), a tardiness cost w_j is deducted from the project reward r_j at the project's completion.

$$d_j \in \{0, 1, 2, \ldots, F_j\} \tag{6.5}$$

The absence of a project type j is shown by a project state $\boldsymbol{P}_j$ where all task states are 0 ($x_{j,i} = 0, \forall i$). For these cases the due date state of type j project is taken as 0 ($d_j = 0$).

$$\boldsymbol{P}_j = (0, 0, \ldots, 0, 0) \tag{6.6}$$

An accepted arrival of a project type j in the previous period is represented by a project state $\boldsymbol{P}_j$ where all task states are -1 ($x_{j,i} = -1, \forall i$) and the due date state's

[2] (6.3) is not a matrix when project types have different task numbers I_j.

value is F_j.

$$\boldsymbol{P}_j = (-1, -1, \ldots, -1, F_j) \tag{6.7}$$

Action

An action a represents the decision of which tasks to begin processing for those task states that are "pending for processing". The action consists of action elements $a_{j,i}$ for each task $i \in I_j$ of all project types ($j \in J$). An action element takes the value of 1 ($a_{j,i} = 1$) to represent the decision to start processing a qualifying task and takes the value 0 ($a_{j,i} = 0$) otherwise.

$$a = \begin{bmatrix} a_{1,1}, & a_{1,2}, & \ldots & a_{1,I_1} \\ a_{2,1}, & a_{2,2}, & \ldots & a_{2,I_2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{J,1}, & a_{J,2}, & \ldots & a_{J,I_J} \end{bmatrix} \tag{6.8}$$

An action a must fulfil three requirements:
(1) The selected tasks for processing must have the task state "pending for processing".

$$a_{j,i} = 1 \Rightarrow x_{j,i} = -1 \text{ for } \forall j \in J, \forall i \in I_j \tag{6.9}$$

(2) There must be enough free resources of each type to allocate for processing the selected tasks.

$$\sum_{j=1}^{J} \sum_{i=1}^{I_j} b_{j,i}^{k} \mathcal{I}\{a_{j,i} = 1\} \leq B_k^{\text{free}}(s) \forall k \in K \tag{6.10}$$

(3) All predecessor tasks of the selected tasks must be completed.

$$a_{j,i} = 1 \Rightarrow x_{j,m} = 0 \text{ for } \forall m \in \mathcal{M}_{j,i}, \forall j \in J, \forall i \in I_j \tag{6.11}$$

Here, $\mathcal{M}_{j,i}$ represents the predecessor task set of a task i from project type j. Elements of $\mathcal{M}_{j,i}$ are tasks required to be completed before task i of project j. m represents a predecessor of task i ($m \in \{1, 2, ..., I_j\} \setminus i, m \in \mathcal{M}_{j,i}$). The action with no task processing decision where all action elements are zero is also a valid action, and it is called as "do not initiate any task". More than one action may fulfil all three requirements for a pre-decision state. The set of these actions is named as an action set $\boldsymbol{A}(s)$.

Post-decision state

A post-decision state $\hat{s}$ represents the system information immediately after a decision epoch and just before the transition time begins. In other words, a post-

decision state is the system information from the pre-decision state s updated by an action a but before any task processing or random event occurs ($\hat{s} := (s, a)$). A post-decision state consists of post-decision project states $\hat{\boldsymbol{P}}_j$. A post-decision project state consists of post-decision task states $\hat{x}_{j,i}$ of each task $i \in I_j$ and the same due date states d_j of the initial project state $\boldsymbol{P}_j$.

$$\hat{s} = \begin{bmatrix} \hat{x}_{1,1}, & \hat{x}_{1,2}, & \dots & \hat{x}_{1,I_1}, & d_1 \\ \hat{x}_{2,1}, & \hat{x}_{2,2}, & \dots & \hat{x}_{2,I_2}, & d_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \hat{x}_{J,1}, & \hat{x}_{J,2}, & \dots & \hat{x}_{J,I_J}, & d_J \end{bmatrix} \tag{6.12}$$

A post-decision task state $\hat{x}_{j,i}$ is the updated status of a task from the preceding pre-decision task state $x_{j,i}$. It is only the states of tasks that have been selected to start their processing ($a_{j,i} = 1$) that change from the pre-decision state of -1 to the post-decision state $t_{j,i}^{max}$.

$$\hat{x}_{j,i} = \begin{cases} t_{j,i}^{max}, & \text{if } a_{j,i} = 1 \\ x_{j,i}, & \text{otherwise} \end{cases} \tag{6.13}$$

Pre-acceptance state

A pre-acceptance state s^* represents the system information at the end of the transition time but immediately before the pre-decision state s' at the following decision epoch. A pre-acceptance state consists of its project states $\boldsymbol{P^*}_j$ which consist of pre-acceptance task states $x_{j,i}^*$ for each task $i \in I_j$ and pre-acceptance due date states d_j^*. A pre-acceptance state shows the task processing progress after a post-decision state during the transitional time without new project arrivals. (6.16) shows possible task state transitions from a $\hat{x}_{j,i}$ to $x_{j,i}^*$ with their probabilities. As (6.14) presents, a pre-acceptance due date state is zero ($d_j^* = 0$) if project type j is completed or the post-decision due date state is zero. For the other possibilities a pre-acceptance due date state is equal to post-decision due date state minus one.

$$d_j^* = \begin{cases} 0, & \text{if } d_j = 0 \\ 0, & \text{if } \hat{x}_{j,i} \geq 0 \text{ for } \exists i \in \{1, ..., I_j\} \text{ and } x_{j,i}^* = 0 \text{ for } \forall i \in \{1, ..., I_j\} \\ d_j - 1, & \text{otherwise} \end{cases} \tag{6.14}$$

If a type j project arrived during the transition time and the j type project state's status $\boldsymbol{P^*}_j$ is completed or empty ($x_{j,i}^* = 0$ for $\forall i \in \{1, ..., I_j\}$), the system accepts the new j type project. Otherwise, the system rejects the new arrival. From a pre-acceptance state s^* to the following pre-decision state s', the new task status

becomes -1 and due date state state becomes F_j.

Transition function

The *transition function* represents the transformation of a system from a post-decision state $\hat{s}$ to a pre-decision state s' at the next decision epoch by random events c_{t+1} during the transition time. The random events include new project arrivals λ_j, task completions $\gamma_{j,i}(\hat{x}_{j,i})$ and project completions.

A project from each type may arrive in the system during a transition time according to its type's arrival probability λ_j. We consider a semi-open project acceptance process where the system only accepts a new arrival of a type j project if no type j project exists in the system, either in processing or waiting. Otherwise, the system rejects the new arrival and continues its processing as if there was no arrival.

A task may complete processing according to a conditional probability $\gamma_{j,i}(\hat{x}_{j,i})$, if the task's processed time following this transition ($t_{j,i}^{max} + 1 - \hat{x}_{j,i}$) is equal to or greater than its minimal possible duration $t_{j,i}^{min}$.

The probability of reaching a pre-decision state s' from a post-decision state $\hat{s}$ with the transition function $P(s'|s,a)$ is the joint probability of task completions $P(x'_{j,i}|\hat{x}_{j,i})$ and project arrivals $P(\boldsymbol{P}'_j|\hat{\boldsymbol{P}}_j)$:

$$P(s'|s,a) = \prod_{j=1}^{J} \left(\prod_{i=1}^{I_j} P(x^*_{j,i}|\hat{x}_{j,i}) \right) P(\boldsymbol{P}'_j|\boldsymbol{P^*}_j) \tag{6.15}$$

$$P(x^*_{j,i}|\hat{x}_{j,i}) = \begin{cases} \gamma_{j,i}(\hat{x}_{j,i}), & \text{if } t_{j,i}^{max} - t_{j,i}^{min} + 1 \geq \hat{x}_{j,i} \geq 1 \text{ and } x^*_{j,i} = 0 \\ 1 - \gamma_{j,i}(\hat{x}_{j,i}), & \text{if } t_{j,i}^{max} - t_{j,i}^{min} + 1 \geq \hat{x}_{j,i} \geq 2 \text{ and } x^*_{j,i} = \hat{x}_{j,i} - 1 \\ 1, & \text{if } t_{j,i}^{max} - t_{j,i}^{min} + 1 < \hat{x}_{j,i} \text{ and } x^*_{j,i} = \hat{x}_{j,i} - 1 \\ 1, & \text{if } x^*_{j,i} = \hat{x}_{j,i} \leq 0 \\ 0, & \text{otherwise} \end{cases} \tag{6.16}$$

$$P(\boldsymbol{P}'_j|\boldsymbol{P^*}_j) = \begin{cases} \lambda_j, & \text{if } x'_{j,i} = -1 \text{ and } x^*_{j,i} = 0 \text{ for } \forall i \in \{1, ..., I_j\} \\ 1 - \lambda_j, & \text{if } x'_{j,i} = x^*_{j,i} = 0 \text{ for } \forall i \in \{1, ..., I_j\} \\ 1, & \text{if } x'_{j,i} = x^*_{j,i} \neq 0 \text{ for } \forall i \in \{1, ..., I_j\} \\ 0, & \text{otherwise} \end{cases} \tag{6.17}$$

Here in (6.16), the first line represents that the post-decision task state of task i from project type j allows for task completion and, with $\gamma_{j,i}(\hat{x}_{j,i})$ probability, the task will be completed by the pre-acceptance state. The second line represents that the post-decision task state of task i from project type j allows task completion and,

with $1-\gamma_{j,i}(\hat{x}_{j,i})$ probability, the task will not be completed by the pre-acceptance state. The third line represents that the post-decision task state of task i from project type j does not allow task completion and, with 100% probability, the task will not be completed by the pre-acceptance state. The fourth line represents that the post-decision task state of task i from project type j is completed or waiting for processing and, with 100% probability, the task will retain its status in the pre-acceptance state.

In (6.17), the first line represents that, with λ_j probability, there will be an arrival of project type j during the transition time and the new type j project will take the place of the previously completed or non-existing type j project. The second line represents that, with $1-\lambda_j$ probability, there will be no new arrival of project type j during the transition time. The third line represents that, with 100% probability, the arrival of projects will not affect the status of on going or waiting projects of the same type as the new project will be rejected.

6.2.4 Objective function

The immediate reward represents the state transition reward from $\hat{s}$ to s'. We consider a project completion reward r_j and a tardiness cost w_j. Thus we use the term *profit* instead of a reward. The immediate profit $R_{s,a,s'}$ is the sum of completed project rewards minus the tardiness cost of late completions ($d_j = 0$).

$$\begin{aligned} R_{s,a,s'} = \sum_{j=1}^{J} \Big\{ & \big(r_j - w_j \mathcal{I}\{d_j = 0\}\big) \\ & \mathcal{I}\left\{\hat{x}_{j,i} > 0 \text{ for } \exists i \in \{1,...,I_j\} \text{ and } x'_{j,i} \leq 0 \text{ for } \forall i \in \{1,...,I_j\}\right\} \Big\} \end{aligned} \tag{6.18}$$

Here, the first indicator is for late project completion, and it takes the value 1 if a project completes later than its due date (i.e., the project's due date state $d_j = 0$) and is 0 otherwise. The second indicator is for project completion and takes the value 1 if a project completes (at least one task is in progress in the post-decision state and all project tasks are complete at the end of the period) and is 0 otherwise.

Our objective function seeks to find the policy that maximises the expected total discounted long-run profit:

$$V^*(s) = \max_{\pi \in \Pi} \mathbb{E}^{\pi} \left[\sum_{t=1}^{\infty} \alpha^{t-1} R_{s_t,a,s_{t+1}} \right] \tag{6.19}$$

Here, $R_{s_t,a,s_{t+1}}$ is the immediate profit of state transition from pre-decision states s_t

to s_{t+1} under the action a at the time t. α is the discount factor between $(0, 1)$[3]. π is a policy from the set of all non-anticipating policies Π that prescribe in every state s an action from the action set $\boldsymbol{A}(s)$.

6.3 Approximate dynamic programming (ADP)

In theory, the problem (6.19) can be solved using the Bellman equation:

$$V^*(s) = \max_{a \in \boldsymbol{A}(s)} \sum_{s' \in \mathcal{S}} P(s'|s, a)[R_{s,a,s'} + \alpha V^*(s')] \tag{6.20}$$

But in practice, it suffers from "the curse of dimensionality". "The curse of dimensionality" means that the number of states and computational requirements expands exponentially with the number of state variable (Sutton and Barto, 2018). Chapter 5 investigated the limitations of DP and stated that a state space larger than their five projects with two tasks problem is computationally intractable for their hardware.

ADP is a modelling strategy to overcome "the curse of dimensionality" problem of DP due to the use of the Bellman equation (Powell, 2009). In our ADP algorithm, we estimate the value function of the Bellman equation (6.20) using a linear approximation model (subsection 6.3.1). The linear approximation model only requires the current state and action information, and future state information and storing the states becomes unnecessary. With the linear approximation model, the decision making is done in an online fashion; thus, ADP can be used for larger size problems.

6.3.1 Linear approximation model

For the approximate value function $\bar{V}(\hat{s})$, we construct a linear approximation model with two state variables as features:

$$\begin{aligned}\bar{V}(\hat{s}) = \sum_{j}^{J} \Bigg\{ \theta_j^1 \sum_{i}^{I} \left\{ \left\{ t_{j,i}^{max} - \hat{x}_{j,i} + 1 \right\} \mathcal{I}\left\{ \hat{x}_{j,i} > 0 \right\} + t_{j,i}^{max} \mathcal{I}\left\{ \hat{x}_{j,i} = 0 \right\} \right\} + \\ \theta_j^2 \sum_{i}^{I} \sum_{k}^{K} \left\{ b_{j,i}^{k} \mathcal{I}\left\{ \hat{x}_{j,i} > 0 \right\} \right\} \Bigg\}\end{aligned} \tag{6.21}$$

The first feature is the number of periods that will have been spent processing each type of project by the end of the current period ($\sum_i^I \left\{ t_{j,i}^{max} - \hat{x}_{j,i} + 1 \right\}$). Here,

[3]Note that the limit $\alpha = 0$ would correspond to the myopic (single-period) profit, while the limit $\alpha = 1$ would correspond to the long-run average profit

$\left\{t_{j,i}^{max} - \hat{x}_{j,i}\right\}$ represents task i's processed time. Since the task's processed time is zero when the action is taken ($\hat{x}_{j,i} = t_{j,i}^{max}$), we use the task's processed time after transition time ends ($t_{j,i}^{max} - \hat{x}_{j,i} + 1$) to differentiate the effect of actions. This feature generates a higher score as the project is processed longer, creating a prioritisation of the ongoing and close to completion project types over new arrivals. So, this feature helps to reduce the lateness of completions.

The second feature is the amount of resource which is allocated to each type of project when the action to begin processing has been taken. We considered the post-decision state resource allocation because it gives the best information about an action's resource requirement (e.g., for single period tasks, the resource allocation information of action may disappear after one transition time. Thus, the post-decision state's resource allocation gives the most precise information about resource usage after the action is taken). This feature generates a higher score when the resource utilisation is higher, which creates a prioritisation of actions with higher resource usages. So more tasks can be processed in one transition period.

Coefficients θ_j^1 and θ_j^2 are generated using simulation training as described in subsection 6.3.2. The simulation training generates the best coefficient values to help our linear approximation model to mimic the discounted long-run profit of simulated project scheduling. The combination of coefficients, the first and second features, creates a balanced approximation function that seeks to generate high profits and to process more projects with less lateness of completions.

These features were chosen according to our comparisons with different approximation functions. In preliminary experiments, we generated eight different approximation functions which are combinations of features including the amount of time spent processing the projects in the system, the amount of resources being used by the projects, the amount of free (unused) resource, the time until the project due date expires and the total remaining processing times across the projects. According to our experiments the 2-feature linear approximation model presented here showed consistently strong performance in comparison to the alternatives across a range different problem settings.

6.3.2 Generation of approximation function coefficients

The ADP algorithm generates coefficients for features in the linear approximation model (6.21) using the least-squares fitting method, which minimises the residual between the linear approximation model results and simulated results. The coefficient generation process is summarised in Algorithm 5.

Algorithm 5 ADP

procedure ADP ALGORITHM

$\forall j \in J, \theta_j^1 = \theta_j^2 = 0$, the initial state $s_1 = \mathbf{0}$. ▷ initial values

for $itr = 1$ to $Iteration$ **do** ▷ for each iteration

$\tilde{V}_{sim} = 0$ ▷ $\tilde{V}_{sim}$ is cumulative simulation profit

for $sim = 1$ to $Simulation$ **do** ▷ for each simulation

$\forall j \in J, D_j^1(sim) = \sum_i^I \Big\{ \{t_{j,i}^{max} - x_{j,i} + 1\} \mathcal{I}\{x_{j,i} > 0\} + \{t_{j,i}^{max}\} \mathcal{I}\{x_{j,i} = 0\} \Big\}$,

$\forall j \in J, D_j^2(sim) = \sum_i^I \Big\{ b_{j,i} \mathcal{I}\{x_{j,i} > 0\} \Big\}$ ▷ $x_{j,i} \in s_1$

for $t = 1$ to $Period$ **do** ▷ for each simulation period

find $\boldsymbol{A}(s_t)$ for s_t ▷ $\boldsymbol{A}(s_t)$ is the action set for s_t

find $a = \underset{a \in \boldsymbol{A}(s_t)}{\arg\max} \sum_j^J \Big\{ \theta_j^1 \sum_i^I \{\{t_{j,i}^{max} - \hat{x}_{j,i} + 1\} \mathcal{I}\{\hat{x}_{j,i} > 0\} + \{t_{j,i}^{max}\} \mathcal{I}\{\hat{x}_{j,i} = 0\}\} +$
$\hookrightarrow \ \theta_j^2 \sum_i^I \sum_k^K \{b_{j,i}^k \mathcal{I}\{\hat{x}_{j,i} > 0\}\} \Big\}$

compute $s_{t+1} = s^M(s_t, a, c_{t+1})$ ▷ state iteration via simulation

$\tilde{V}_{sim} \leftarrow \tilde{V}_{sim} + \alpha^{t-1} R_{s_t, a, s_{t+1}}$ ▷ $R_{s_t,a,s_{t+1}}$ explained at subsection 6.2.4

end for

$s_1 \leftarrow s_{Period+1}$ ▷ initial state for the next simulation

end for

$\forall j \in J, \theta\_new_j^1 = \theta\_new_j^2 = 0$ ▷ initial new coefficients

$\underset{\theta\_new_j^1, \theta\_new_j^2}{\arg\min} \sum_{sim=1}^{Simulation} \Big\{ \tilde{V}_{sim} - \sum_j^J \big(\theta\_new_j^1 D_j^1(sim) + \theta\_new_j^2 D_j^2(sim) \big) \Big\}^2$

$\hookrightarrow$ ▷ The linear approximation of the covariance matrix

$\tau = \tau_{harmonic} / (\tau_{harmonic} + itr - 1)$ ▷ harmonic step size

$\forall j \in J, \theta\_old_j^1 = \theta_j^1$ and $\theta\_old_j^2 = \theta_j^2$

$\forall j \in J, \theta_j^1 = (1 - \tau)\theta\_old_j^1 + \tau\theta\_new_j^1$

$\forall j \in J, \theta_j^2 = (1 - \tau)\theta\_old_j^2 + \tau\theta\_new_j^2$

end for

return $\forall j \in J$, θ_j^1 and θ_j^2

end procedure

Here, we train our approximation function with a set amount of iterations. In the first iteration, we assume the initial pre-decision state is an empty state with no existing project, and coefficients are zero. In each iteration, we run a set amount of simulations, and from each simulation, we collect features D_j^1 and D_j^2 of the initial pre-decision states and cumulative simulated profit (CSP). After simulations are completed, we estimate coefficients ($\forall j \in 1 : J \quad \theta_j^1, \theta_j^2$) by minimizing the sum of the squared deviations between the cumulative discounted profits and the linear approximation model (6.21) using a linear least-squares regression method. We call estimated coefficients as θ_new and existing coefficients as θ_old, and we use them in a dynamic step-size function (6.22) to generate coefficients of the new iteration.

$$\forall j \in J, \forall h \in 1 : 2, \theta_j^h = (1 - \tau)\theta\_old_j^h + \tau\theta\_new_j^h \tag{6.22}$$

We generate the dynamic step-size value τ with the harmonic step-size method

(6.23). We set harmonic step-size value as $\tau_{harmonic} = Iteration^{0.5}$. [4]

$$\tau = \tau_{harmonic}/(\tau_{harmonic} + itr - 1) \tag{6.23}$$

We use the terminal pre-decision state of the iteration as the initial pre-decision state in the new iteration. After a specific amount of iterations, the final coefficients are used in the linear approximation model for online decision making.

During a simulation, we find the action set $\boldsymbol{A}(s)$ of the current pre-decision state s_t and, select the most profitable action using the objective function in (6.24). If multiple actions have the highest profits, we randomly select among them.

$$\begin{aligned} a = \underset{a \in \boldsymbol{A}(s_t)}{\arg\max} \sum_{j}^{J} \Big\{ \theta_j^1 \sum_{i}^{I} \big\{ \{t_{j,i}^{max} - \hat{x}_{j,i} + 1\} \mathcal{I}\{\hat{x}_{j,i} > 0\} + \{t_{j,i}^{max}\} \mathcal{I}\{\hat{x}_{j,i} = 0\} \big\} + \\ \theta_j^2 \sum_{i}^{I} \sum_{k}^{K} \big\{ b_{j,i}^{k} \mathcal{I}\{\hat{x}_{j,i} > 0\} \big\} \Big\} \end{aligned} \tag{6.24}$$

The post-decision state $\hat{s}$ begins with the implementation of the best action. Then random events c_{t+1} are simulated according to the transition function for during the transition time and the new pre-decision state s' is achieved. If any project completes during the transition time, their profit is added to the cumulative profit with discounting using the discounting function α^{t-1}. This simulation process repeats for a specific amount of periods, and the final pre-decision state is used as an initial pre-decision state in the next simulation.

6.3.3 Online decision making

The approximation function evaluated by Algorithm 5 can be used for online decision making for any pre-decision state. Online decision making process is summarised in Algorithm 6. First, all actions for the pre-decision state are generated, then expected profits are calculated using the approximation function for each action. The action with the highest profit is used for the state. If multiple actions have the highest profits, one of them is selected randomly.

6.4 Compared Algorithms

We used DP, ORBA, GA and RBA for benchmarking with our ADP algorithm. ORBA and GA are applied to the dynamic problem using a reactive scheduling

[4] We chose the step-size value according to Powell (2011). We also tried KESTEN's Stepsize Rule but we received better coefficient values with the harmonic step-size in our tests.

Algorithm 6 Online Decision Making

procedure ONLINE DECISION MAKING(ADP)
find $\boldsymbol{A}(s_t)$ for s_t ▷ $\boldsymbol{A}(s_t)$ is the action set for s_t
find $\tilde{a} = \underset{a \in \boldsymbol{A}(s_t)}{\arg\max} \Big\{ \sum_{s'_t \in \mathcal{S}} P(s'_t|s_t, a) R_{\hat{s}_t} + \alpha^{t-1} \sum_j^J \Big\{ \theta_j^1 \sum_i^I \Big\{ \{t_{j,i}^{max} - \hat{x}_{j,i} + 1\}$
$\hookrightarrow \mathcal{I}\{\hat{x}_{j,i} > 0\} + \{t_{j,i}^{max}\}\mathcal{I}\{\hat{x}_{j,i} = 0\} \Big\} + \theta_j^2 \sum_i^I \sum_k^K \Big\{ b_{j,i}^k \mathcal{I}\{\hat{x}_{j,i} > 0\} \Big\} \Big\} \Big\}$
select randomly a $\pi(s_t) \in \Pi^*$ ▷ Π^* is the set of all best action for s_t
return $\tilde{a}$
end procedure

method which reschedules the plan when a new project arrival occurs by rerunning the algorithm. ORBA and GA generate a task processing order to create an action for a given pre-decision state. DP generates the optimal policy which we denote π^*. RBA directly creates an action for a pre-decision state according to some predefined rules or criteria.

6.4.1 Dynamic programming (DP)

DP calculates optimal policies from an MDP model of the problem by solving the Bellman equation (Sutton and Barto, 2018). We used the value iteration method. We used DP only for problems from Chapter 5 for benchmarking.

Algorithm 7 value iteration

procedure STATE VALUE ITERATION PROCEDURE
$\beta = 0.001$ ▷ β is the stopping parameter
For each $\forall s \in \mathcal{S}, V^{old}(s) = 0$ ▷ initial state values
do
for $\forall s \in \mathcal{S}$ **do**
$V(s) = \max_{a \in A(s)} \sum_{s' \in \mathcal{S}} p(s'|s, a)(R_{s_t, a, s_{t+1}} + \alpha V^{old}(s'))$ ▷ value function
end for
$W_{max} = \max_{s \in \mathcal{S}} [V(s) - V^{old}(s)]$ ▷ maximum value change
Update $\forall s \in \mathcal{S}, V^{old}(s) = V(s)$
while $W_{max} > \beta(1 - \alpha)/(2\alpha)$
end procedure

6.4.2 Optimal reactive baseline algorithm (ORBA)

ORBA is an exact algorithm for the static RCMPSP and does not consider new project arrivals. ORBA is a brute force algorithm; it calculates the reward and makespan of every feasible task scheduling order (TSO) of any waiting for

processing tasks and finds the most profitable TSO. The TSO is converted to a non-idling action for the current pre-decision state. A non-idling action is one that will always allocate resource to tasks when it is possible to do so. ORBA generates the best possible result of a reactive scheduling method. The ORBA used here extends that in Chapter 5 by allowing for multiple resource types.

ORBA is presented in Algorithm 8. For a given pre-decision state s_t ORBA selects the feasible TSO of maximal profit. In the case of ties the candidate schedule with minimal makespan is selected, and the schedule of smallest project and task indices selected from any remaining candidate schedules.

Algorithm 8 ORBA

procedure
Y_t is the set of tasks $(j,i) \in$ state s_t for which $x_{j,i} = -1$.
ϵ_{Y_t} is the set of all feasible permutations of the set Y_t.
A feasible TSO is a permutation σ of Y_t such that, for any $m < n$ with $\sigma(m) = (j,i)$ and $\sigma(n) = (j,k)$, $k \notin \mathcal{M}_{j,i}$.
$V_{max} = 0$, $makespan_{min} = \inf$, $TSO^* = \emptyset$ ▷ initial values
for $h = 1$ to $|Y_t|!$ **do** ▷ for all permutations of Y_t
if $\epsilon_{Y_t}(h)$ is a feasible TSO **then**
evaluate $\tilde{V}_{sim}(\epsilon_{Y_t}(h))$ and $makespan_{sim}(\epsilon_{Y_t}(h))$
if $(\tilde{V}_{sim}(\epsilon_{Y_t}(h)) \geq V_{max})$ **or** $(\tilde{V}_{sim}(\epsilon_{Y_t}(h)) = V_{max}$ **and** $makespan_{sim}(\epsilon_{Y_t}(h)) < makespan_{min})$ **then**
$TSO^* = \epsilon_{Y_t}(h)$
$V_{max} = \tilde{V}_{sim}(\epsilon_{Y_t}(h))$
$makespan_{min} = makespan_{sim}(\epsilon_{Y_t}(h))$
end if
end if
end for
return TSO^*
end procedure

Here, Y_t is the set of all waiting for processing tasks ($x_{j,i} = -1$) for a given pre-decision state s_t. ϵ_{Y_t} is the set of all feasible permutations of the set Y_t. $\tilde{V}_{sim}(*)$ is the cumulative profit of TSO. $makespan_{sim}(*)$ is the makespan of TSO. TSO^* is the best TSO found so far. V_{max} is the highest profit found among the feasible TSOs. $makespan_{min}$ is the shortest makespan found among the feasible TSOs.

The generated TSO is converted to non-idling action a for given pre-decision state s_t using a serial scheduling generation scheme (SGS). In a SGS, if there are enough free resources to process the first task in the TSO, its action becomes one ($a_{j,i} = 1$) and its resource usage requirements are subtracted from the free resources. The process then repeats for the remaining tasks in order of the TSO. The algorithm stops when all waiting for processing tasks have been considered

and an non-idling action is produced.

The TSO for the remaining tasks can be used to create future actions for the following periods as long as no new project arrives. If a new project arrival disturbs the system, the current TSO becomes invalid, and ORBA generates a new TSO.

ORBA runs in factorial time. Due to the huge computation time requirement of brute force algorithms, only small size dynamic and stochastic RCMPSP problems can be solved. Thus we limit our test problems with ORBA to a maximum of 10 tasks.

6.4.3 Genetic algorithm (GA)

GA is a heuristic algorithm which searches the solution space using a set of solutions (population). GA then improves the population many times (generation) using bio-inspired operators such as crossover and mutation to find better solutions. We used the genetic algorithm to benchmark with our ADP algorithm since GA is the most popular method for RCPSP family. GA is applied to dynamic problems using a reactive scheduling method. We used GA from Chapter 5 and, in this chapter, we extended it to multiple resource types.

For a given pre-decision state s_t, GA generates the desired population size amount of feasible TSOs, which are random permutations of the waiting for processing tasks ($x_{j,i} = -1$). The algorithm evaluates profits and makespans of TSOs and ranks the TSOs using these values. TSOs with higher profits get a higher rank. In the case of ties, TSOs with smaller makespans gets higher ranks. If the tie continues, TSOs ranked according to their creation time (earliest to latest). This first set of TSOs is called the first generation, and the highest ranked TSO is called as the best TSO. After the first generation is generated, GA begins iterations.

At the each iteration, GA creates an empty set of TSOs and fills this new set with TSOs to the desired population size amount using elitist selection, crossover and mutation operators. The elitist selection operator copies the desired amount of highest ranked TSOs from the previous generation of TSOs to the new empty set of TSOs (new generation). Crossover and mutation operators fills the rest of the new generation. The crossover operator randomly selects two TSOs from the previous generation and randomly selects a task inside of the first TSO. The scheduling order of tasks from first task [5] Then, the crossover operator copies the rest of tasks of the first (selected) TSO to the new TSO (to after the randomly selected task) but changes the order of these tasks according to order of these

[5]The first task represent the first(earliest) to be processed task. to selected task is copied from the first (selected) TSO, to (make) a new TSO.

tasks in the second (selected) TSO. The new TSO is always a feasible TSO, since it is created according to order of tasks in both selected feasible TSOs. The new TSO may be adjusted by the mutation operator with a desired probability or it is added to the new generation. Under the mutation operation a task is selected at random and the location of this task in the TSO is randomly re-assigned. The new location can not be later than task's previous order and can not be sooner than its latest to be processed predecessor task. Thus the mutation operator also ensures that the newly generated TSO is a feasible TSOs. Then the new TSO is added to the new generation. When size of the new generation reaches to the population size amount, TSOs are ranked same as in the first generation. GA iterates the generations till the desired number of generation is reached. The best TSO of the final generation is used for decision making.

The TSO is converted to an action in the same way as for ORBA. Similar to ORBA, GA's TSO can be used for future pre-decision states as long as a new project arrival does not disturb the system.

Since the reactive scheduling method reruns GA for each time an arrival disturbs the processing plan, the computational time requirement increases with the problem size.

6.4.4 Rule based algorithm (RBA)

Rule-Based Algorithm (RBA) is a priority-based heuristic algorithm which uses the longest task first priority rule. We considered RBA in benchmarking to show the performance of a simple heuristic algorithm. Due to the simplicity of the algorithm, it runs very fast for all problem sizes.

For a given pre-decision state s_t, RBA creates a feasible TSO where the tasks with the longest task processing durations have priority against other tasks. Then the TSO is converted to an action as same as in ORBA. In contrast to ORBA and GA, RBA does not use the same TSO for future pre-decision states, since RBA generates a new TSO faster than recycling the TSO.

6.5 Computational Results

We simulate the dynamic project scheduling environment with random new project arrivals and stochastic task durations, and we compare the expected discounted long-run profit performance of DP, ADP, ORBA, GA and RBA. Algorithm 9 shows the simulation procedure we used in our comparisons. The statistical significance of ADP against other methods are shown in the tables at

Algorithm 9 Simulation

procedure PROFIT SIMULATION
  $Simulation = 100, Period = 1000.$ ▷ initial values
  for $sim = 1$ to $Simulation$ **do** ▷ for each simulation
    $\tilde{V}_{sim} = 0$ ▷ $\tilde{V}_{sim}$ is cumulative simulation profit
    $s_1 = \mathbf{0}$ ▷ the initial empty pre-decision state
    for $t = 1$ to $Period$ **do** ▷ for each simulation period
      $a \in \pi$ ▷ π is the policy of selected solution method
      $s_{t+1} = H(s_t, a)$ ▷ $H(s_t, a)$ is state iteration using simulation
      $\tilde{V}_{sim} += \alpha^{t-1} R_{s_t, a, s_{t+1}}$
    end for
  end for
  $\tilde{V} = \frac{1}{Simulation} \sum_{sim=1}^{Simulation} \tilde{V}_{sim}$
  return $\tilde{V}$
end procedure

three levels $(0.001, 0.01, 0.05)$. All other statistical comparisons described in the text are taken at a significance level of 0.05 but at not reported in the chapter.

We used 100 problem scenarios in our comparison which are combination of 10 project settings and 10 project arrival probabilities. These arrival probabilities λ_j are 0.01 and from 0.1 to 0.9 with increment of 0.1. Since we considered a dynamic environment $\lambda_j = 0$ is not used in this comparison instead $\lambda_j = 0.01$ is used to represent the nearly-static case. In 5 of 10 problem scenarios, results are not statistically significantly different for $\lambda_j = 0.01$, which shows that all the algorithms are well tuned for the nearly-static case. Also $\lambda_j = 1$ is not used because it causes a non-ergodic MDP where some feasible states can not be reachable from any states (for example a state where all projects are completed but no new project has arrived).

ADP, GA and RBA are compared on all of the 100 scenarios but DP and ORBA only used in comparison on 30 smaller size problems. ADP and the compared algorithms are coded in JuliaPro 1.3.1.2. All tests are performed on a desktop computer with Intel i5-6500T CPU with 2.50 GHZ clock speed and 32 GB of RAM.

Here, we run 100 simulations[6] for 1000 simulation periods with an $\alpha = 0.999$ discount rate. If a period represent a day, this period would represent approximately three years of processing time. All simulations start from an empty pre-decision state ($s_1 = \mathbf{0}$). In each simulation period an action is generated with the policy being investigated (π). Then the following pre-decision state is generated given the action taken and the transition function. The profit is then is generated, recorded and included within $\tilde{V}_{sim}$. The discounted profits of completed projects during the state iteration are added to $\tilde{V}_{sim}$. After the end of the all simulations, the average discounted long-run profit $\tilde{V}$ of the investigated

[6] 16 simulations for GA.

solution method is calculated.

ADP (Algorithm 5) is trained for 100 iterations each having 100 simulations with 1000 periods. GA is trained for 100 generations, each with 100 populations. The elitest selection operator transfers the best 10% of the population to the next generation. A new TSO created by the crossover operator is handled by the mutation operator with a 50% chance.

While ADP requires training once prior to the simulations, GA requires multiple training occurrences during the simulations. GA is required to generate a new schedule each time a new project arrives. In these settings, despite the fact that GA generates a baseline schedule very fast, the total computation time of all GA trainings is much higher than single training of ADP. For balanced comparison time of the algorithms, we run 16 simulations for GA and 100 simulations for other methods.

In this chapter, we assume that number of tasks of different project types are equal and all tasks have a completion duration range $t^{max} - t^{min} + 1 = 3$ (outwith one exception where this duration is 2). (6.25) shows the completion probabilities used in this chapter.

$$\gamma_{j,i}(\hat{x}_{j,i}) = \begin{cases} \frac{1}{\hat{x}_{j,i}}, & \text{if } t^{max}_{j,i} = t^{min}_{j,i} + 2 > 2, t^{max}_{j,i} - \hat{x}_{j,i} \geq t^{min}_{j,i} - 1 \\ \frac{1}{3}, & \text{if } t^{max}_{j,i} = 2 = t^{min}_{j,i} + 1, \hat{x}_{j,i} = 2 \\ 1, & \text{if } t^{max}_{j,i} = 2 = t^{min}_{j,i} + 1, \hat{x}_{j,i} = 1 \\ 1, & \text{if } t^{max}_{j,i} = 1 = t^{min}_{j,i}, \hat{x}_{j,i} = 1 \\ 0, & \text{otherwise} \end{cases} \tag{6.25}$$

6.5.1 Comparison to optimal

We used project settings from Chapter 5. The problems' data is available in Chapter 5 and at https://github.com/ugursatic/DSRCMPSP. These problems are arbitrarily created to be small and solvable by DP. Using these small size problems we able to compare ADP's performance with optimal policies of DP, scheduling orders of ORBA, GA and RBA. The results of compared algorithms are not statistically significantly different from each other in these problems for $\lambda_j = 0.01$. Thus we did not include the comparison results at $\lambda_j = 0.01$ to our results discussion below.

Table 6.1,Table 6.2 and Table 6.3 illustrate discounted long-run profits (which are averages of simulations) of policy generation methods (vertical) at different arrival probabilities (horizontal). The colour of cells show the statistical significance of compared algorithms against ADP.

The simulation results of two projects types, two tasks and one resource type problem is shown in Table 6.1. The optimal results of DP is statistically significantly the most profitable. ADP produces statistically significantly higher profits than GA and RBA, and ADP's profits are statistically significantly higher than ORBA from $\lambda_j = 0.2$ to $\lambda_j = 0.9$. ORBA's results are statistically significantly better than GA from $\lambda_j = 0.2$ to $\lambda_j = 0.9$. RBA produces statistically significantly the lowest results than compared methods.

The simulation results of two projects types, three tasks and one resource type problem is shown in Table 6.2. Profits of DP are statistically significantly higher than ADP. GA and ORBA are the second and third most profitable methods, but their results are not statistically significantly different from each other. ADP's profits are statistically significantly lower than ORBA at $\lambda_j = \{0.4, 0.5, 0.7\}$, but they are statistically significantly higher than RBA at $\lambda_j = 0.9$. ADP's results are not statistically significantly different than GA, ORBA or RBA for the other arrival probabilities.

The simulation results of three projects types, two tasks and one resource type problem is shown at Table 6.3. DP is the statistically significantly most profitable method for 5 of λ_j values and the difference between best values and DP's results are not statistically significant at the other 5 of λ_j values. ORBA and GA are the second and third most profitable methods, but their results are not statistically significantly different then each other except for $\lambda_j = 0.4$. ADP produces the forth highest profits which are statistically significantly higher than RBA from $\lambda_j = 0.2$ to $\lambda_j = 0.9$, and they are statistically significantly higher than ORBA at $\lambda_j = 0.1$ and $\lambda_j = 0.8$.

We see the same result as in Chapter 5 in that the reactive scheduling methods ORBA and GA have close to optimal profits at $\lambda_j = 0.01$ where the results are not statistically significantly different from each other. However, their results usually diverge from optimum as λ_j increases.

The small problem sizes allowed us to investigate why ADP's and ORBA's profits are below DP's. For this purpose, we investigated all 1508 states of the two project types and two tasks problem at $\lambda_j = 0.7$ and compared policies of DP, ADP and ORBA actions by actions. In this problem, the second project's reward and tardiness costs are high. Because of this, the normal completion of the second project is more rewarding than the first project's normal completion, but the late completion of the second project is less rewarding than the late completion of the first project. Furthermore, processing the second task of project type two is a bottleneck for this problem since it requires all resource capacity, while task one of project type two can be processed concurrently with any task of project type

Table 6.1: two project types and two tasks problem.

λ_j	0.01	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
DP	76	507	769	901	1004	1066	1108	1139	1167	1198
ADP	76	469	713	839	910	955	1000	1020	1042	1061
ORBA	73	463	666	777	812	842	853	845	843	829
GA	79	446	638	719	731	744	741	728	722	737
RBA	73	418	521	549	542	523	504	493	480	474

Table 6.2: Two project and three tasks problem

λ_j	0.01	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
DP	119	592	804	908	972	1014	1049	1067	1082	1099
ADP	116	571	761	856	904	937	959	962	978	989
ORBA	119	577	764	863	913	951	968	981	986	996
GA	124	575	777	872	914	949	982	979	970	994
RBA	118	572	766	850	903	934	953	970	981	977

Table 6.3: Three project and two tasks problem

λ_j	0.01	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
DP	189	791	885	941	1019	1101	1190	1255	1340	1384
ADP	184	748	849	899	950	1020	1086	1166	1335	1370
ORBA	192	731	850	951	1050	1135	1208	1274	1329	1366
GA	186	728	849	941	1032	1125	1204	1274	1329	1367
RBA	181	741	814	861	926	991	1057	1123	1175	1230

Significantly lower results than ADP	$p<0.05$,	$p<0.01$,	$p<0.001$
Significantly higher results than ADP	$p<0.05$,	$p<0.01$,	$p<0.001$

one. DP takes different actions than ADP and ORBA at 109 of 1508 states. In these states, the first task of project type two is on processing while project type one is waiting or stalled. DP chooses the "do not initiate any task" action and does not process project type one during a period that would not cause its late completion. This idling period allows DP to process the second task of type two earlier which leads a higher project completion profits. Normal completion of project type two is more profitable and early completion allows the system to accept possible new arrivals of project type two sooner which may, in turn help the system process more type two projects. ADP and ORBA are not able to identify that idling period, thus they process project type one immediately for all of the considered 109 states.

In summary, our comparison on project settings from Chapter 5 shows that ADP can not match the optimal policy of DP. Still, ADP's and DP's results were not statistically significantly different in 4 of 30 problem scenarios. ADP generated statistically significantly better profits than ORBA in 10 of 30 problem scenarios. However, ORBA generated statistically significantly higher profits in 8 problem scenarios, and the results between ADP and ORBA are not statistically significantly

different in 12 problem scenarios. ADP produced statistically significantly higher profits than GA in 9 of 30 problem scenarios; GA's profit was statistically significantly higher at 5 problem scenarios, and there was no statistically significant difference in the remaining 16 problem scenarios. ADP produces statistically significantly higher profits than RBA in 18 problem scenarios, and results between ADP and RBA are not statistically significantly different in 12 problem scenarios. These results show that overall performances of ADP and ORBA are similar, and they are slightly better than GA and much better than RBA.

6.5.2 Test problem generation

The problems of Chapter 5 were the only dynamic and stochastic RCMPSPs in the literature that have a reward after completion, a tardiness cost after a given due date, arrival probability of new projects during a transition time, randomly early, normal and late task completions. However, Chapter 5 only considered small size problems where the project network is sequential (serial, $OS_j = 1$). Thus we generate larger size test problems using ProGen/Max and MPSPLIB problems.

ProGen/Max is an RCMPSP generation software which is developed by Schwindt (1998) which extends its predecessor ProGen (Kolisch et al. (1995)) with an option to consider the minimum and maximum time lags between the start of activities.

We used ProGen/Max to generate RCPSPs with different activity-on-node networks, order strength, task durations, resource usage, resource availability. We combined these RCPSPs problems and added stochastic task completion, project arrival probability, project completion reward, late completion cost, and due date. We add reasonable completion rewards and tardiness costs to each project. We used the generated task durations as expected task durations $t_{j,i}$ and added one minimal possible ($t_{j,i}^{min} = t_{j,i} - 1$) and one maximal possible ($t_{j,i}^{max} = t_{j,i} + 1$) duration options. We tested all problems with ten different Λ_j options which are $0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9$. We generated due date of the project as (6.26) where ρ is an arbitrary factor, and its value is 1.5. We adjusted the resource availability using (6.27) because the combination of resource availabilities of multiple single project problems makes the multi-project problem resource-rich.

$$F_j \approx \left((1 - OS_j) \max \left\{ J \frac{\sum_{k=1}^{K} \left(\frac{\sum_{i=1}^{I_j} (t_{j,i} b_{j,i}^k)}{B_k} \right)}{K}, \max\{t_{j,1}, t_{j,2}, ..., t_{j,I_j}\} \right\} + (OS_j) \max \left\{ \sum_{i=1}^{I_j} t_{j,i}, J \frac{\sum_{k=1}^{K} \left(\frac{\sum_{i=1}^{I_j} (t_{j,i} b_{j,i}^k)}{B_k} \right)}{K} \right\} \right) \rho \tag{6.26}$$

$$B_k \approx \sum_{j=1}^{J} \left(B_k^j \left(\frac{50 - 4J}{100} \right) \right) \tag{6.27}$$

The MPSPLIB (http://www.mpsplib.com/) is a RCMPSP library which contains the problem set of Homberger (2007). These RCMPSP problems are made by combining single project problems of PSPLIB (http://www.om-db.wi.tum.de/psplib) which is RCPSP library (Kolisch and Sprecher (1997)). PSPLIB problems are generated with ProGen.

The MPSPLIB contains 140 instances that differ by project type number, project number, task number, global resource type number and arrival times. Global resources are shared among all projects, and local resources are only used for a single project. Compare to our ProGen/Max generated problems, MPSPLIB problems have predefined tardiness costs and due dates. However, these due dates ($Best_j$) are the shortest completion time found for single project problems, which we need to modify to use in the dynamic multi-project setting.

From the MPSPLIB, we only considered 30 tasks per project problems for algorithm benchmarking. We only use global resources and convert local resources to global ones using (6.27). We use the predefined tardiness costs and twice the tardiness cost as completion rewards to each project. We use the stochastic task completion and arrival probabilities as same as ProGen/Max generated problems. We generated due date of the project as (6.28).

$$F_j \approx Best_j + J \frac{\sum_{k=1}^{K} \left(\frac{\sum_{i=1}^{I_j} (t_{j,i} b_{j,i}^k)}{B_k} \right)}{K} \tag{6.28}$$

6.5.3 Larger size problems

We created 5 project settings with ProGen/Max and 2 project settings with MPSPLIB problems. Detailed information about our problems and more detailed

test results are available at https://github.com/ugursatic/DSRCMPSP. The size of these problems exceed the computational limits of DP and ORBA on our hardware, so we only compared ADP, GA and RBA.

In five project types, five tasks and four resource types problem, small projects are more profitable, and long projects are less profitable. Hence, RBA with longer tasks first rule is disadvantageous in this problem, we expect that ADP outperforms RBA. Table 6.4 shows that while ADP produces statistically significantly better results than RBA in all λ_j values and ADP's results are statistically significantly better than GA at $\lambda_j = 0.01$. GA produces statistically significantly better results than ADP in 7 of 10 λ_j values.

In two project types, ten tasks and two resource types problem, any task of project type one has a longer duration than any task of project type two. Also, project type one's ratio of reward to sum of task durations is four times more than project type two's. Thus this problem is very advantageous for longer task first priority rule of RBA. However, Table 6.5 shows that ADP usually produces the statistically significantly highest profits except for $\lambda_j = 0.01$. GA produces statistically significantly better profits than RBA for most of the λ_j values, but GA and RBA's results are not statistically significantly different from each other for $\lambda_j = \{0.01, 0.7, 0.8\}$.

Five project types, ten tasks and two resource types problem has equal rewards and tardiness costs per task processing time. Thus high resource utilisation is more profitable than project prioritising. Table 6.6 shows that ADP's profits are statistically significantly higher than GA and RBA from $\lambda_j = 0.1$ to $\lambda_j = 0.9$. RBA produce statistically significantly higher profits than GA for 7 of 10 λ_j values, and their result are not statistically significantly different from each other at rest of the λ_j values.

Six projects types, five tasks and two resource types problem consists of six copies of the same project with different reward and tardiness cost combinations. Table 6.7 ADP produces statistically significantly best profit for all arrival probabilities. Then GA produces the second highest except for $\lambda_j = 0.01$ where there is no statistically significant difference between GA and RBA.

Ten project types, ten tasks and two resource types problem is the biggest problem we generated with ProGen/Max. In this problem, we randomly assigned rewards and tardiness costs. Table 6.8 shows that ADP is statistically significantly the most profitable method for all λ_j values. Then GA and RBA produces the second and third highest profits but their results are not statistically significantly different from each other.

Two project types, thirty tasks and four resource types problem is the smallest

MPSPLIB problem in our comparison. The problem name is mp_j30_a2_nr2_set in MPSPLIB, and it has one global and three local resources. We converted the local resources to global using (6.28). Table 6.9 shows that ADP leads to statistically significantly higher profits than GA at all arrival probabilities except for $\lambda_j = 0.01$. ADP's profits are statistically significantly higher than RBA at $\lambda_j = 0.01$ and $\lambda_j = 0.1$ but they are statistically significantly lower than RBA at $\lambda_j = 0.6$ and $\lambda_j = 0.9$

Five project types, thirty tasks and four resource types problem is the biggest problem, we have used in our comparison. The problem name is mp_j30_a5_nr4_set in MPSPLIB. The original problem has three global (types 1, 2 and 3) and one local resource (type 4) types. We combined the local resources of each project to make one global resource. We used the given due dates in the problem without changing them. Combining the resources of single project problems as global resources without reducing their amount made this problem resource rich. Thus the difference between algorithms are not statistically significant. Table 6.10 shows that ADP, RBA and GA produces similar profits with no statistically significant difference except for $\lambda_j = 0.6$ where ADP's result is statistically significant lower than GA.

In summary, our comparison of larger size problems shows that ADP's profit was statistically significantly higher than GA in 48 of 70 problem scenarios; GA's profit was statistically significantly higher for 8 problem scenarios, and there was no statistically significant difference in the remaining 14 problem scenarios. ADP generated statistically significantly higher profits than RBA in 51 problem scenarios; RBA's profit was statistically significantly higher than ADP only in 2 problem scenarios, and results between ADP and RBA are not statistically significantly different in 17 problem scenarios. These results show that the overall performance of ADP is statistically significantly better in the majority of bigger size problems than the GA and RBA.

6.6 Conclusion

Chapter summary. In this chapter, we modelled the dynamic and stochastic RCMPSP as an infinite-horizon discrete-time MDP where projects have identical arrival probabilities at each transition time and tasks have uniformly distributed completion probabilities. The objective of problem is maximising the expected total discounted long-run profit. We used a linear approximation model to design a practical scheduling policy and showed that it performs near-optimally in small problems and compares favourably to existing heuristics in large problems.

The motivation of this study is to create a more comprehensive project schedul-

Table 6.4: Five project types, five tasks and four resource types problem

λ_j	0.01	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
ADP	919	1013	971	965	979	972	974	1086	1095	1128
GA	828	1142	1110	1129	1150	1138	1155	1121	1166	1125
RBA	689	593	573	557	564	546	551	541	520	512

Table 6.5: Two project types, ten tasks and two resource types problem

λ_j	0.01	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
ADP	321	471	492	508	518	518	538	535	542	555
GA	297	453	454	458	460	469	458	452	453	461
RBA	297	424	426	432	436	445	448	456	464	471

Table 6.6: Five project types, ten tasks and two resource types problem

λ_j	0.01	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
ADP	1711	2403	2408	2406	2444	2424	2417	2445	2431	2420
GA	1740	2333	2335	2338	2345	2348	2342	2352	2352	2353
RBA	1724	2335	2364	2368	2373	2391	2381	2385	2391	2392

Table 6.7: Six project types, five tasks and two resource types problem

λ_j	0.01	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
ADP	1306	1854	1854	2098	2124	2139	2149	2157	1876	2051
GA	1189	1307	1297	1320	1307	1314	1316	1298	1327	1341
RBA	1183	1156	1102	1106	1101	1105	1097	1089	1111	1073

Table 6.8: Ten project types, ten tasks and two resource types problem

λ_j	0.01	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
ADP	674	1019	1032	927	794	857	870	858	746	973
GA	521	549	541	547	561	536	538	550	526	551
RBA	518	550	557	553	562	554	548	552	550	561

Table 6.9: Two project types, thirty tasks and four resource types problem

λ_j	0.01	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
ADP	172	208	207	208	208	209	204	208	207	204
GA	165	200	192	189	183	192	192	184	190	193
RBA	157	199	208	209	209	211	208	211	209	209

Table 6.10: Five project types, thirty tasks and four resource types problem

λ_j	0.01	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
ADP	474	1088	1170	1221	1246	1245	1249	1265	1269	1272
GA	505	1104	1175	1193	1230	1250	1279	1244	1269	1277
RBA	485	1094	1178	1217	1238	1260	1254	1264	1274	1280

ing model by considering the uncertainties of stochastic task durations, random new project arrivals, multiple types of resource usages and bigger and complex project networks. For this purpose we suggest a linear approximation model which generates decisions based on resource consumptions and processed times of projects. Our linear approximation model generated the best profits after the exact methods in our comparisons and contributed to the literature by extending the work of Chapter 5 which only considered small-sized projects with sequential networks and single resource type.

Our results show that DP produces statistically significantly better results than ADP for small size problems. However, it suffers from the curse of dimensionality and not suitable for bigger size problems. ORBA often produced equally high profits as our ADP algorithm in the small size problems. However, ORBA runs in factorial time, and it can not be used in bigger size problems. ADP is the best solution method for the bigger size problems in our comparisons. ADP produced statistically significantly higher profits than RBA and GA in majority of problems, and ADP is outperformed by GA or RBA only in a fraction of test problems.

Managerial insights. This study provides an efficient ADP algorithm for dynamic and stochastic RCMPSP that generates statistically significantly higher or equal profits in 88% our comparisons to alternative methods (GA and RBA). Also, this study gives insights to project managers to determine more suitable methods for their environment by providing a performance comparison of ADP, DP, ORBA, GA and RBA methods in various project settings and various arrival probabilities. We suggest using DP for problems that are smaller than the computational limitations of DP, such as two projects and four tasks. However, we suggest using ADP methods for bigger size problems since our test results showed that ADP is usually statistically significantly more profitable than RBA and GA.

Future research direction. In real life, there are more dynamics and stochastic elements in dynamic project scheduling environments than those (stochastic task durations, uncertain new project arrivals, finish to start project networks, multiple resource type and usage) considered in this chapter. Future work might consider other elements of the dynamic project scheduling environment, such as stochastic resource availability or multiple modes of task processing.

Acknowledgments

We acknowledge Mahshid Salemi Parizi for making their code available. The first author acknowledges the Ministry of National Education of The Republic of Turkey for providing a PhD scholarship.

Chapter 7

Conclusion

7.1 Thesis summary

The project execution is frequently gets affected by many uncertainties. For example, the random arrival of new projects makes the previous schedule invalid and uncertain task durations makes the project schedules unreliable. Due to these interruptions, the planned delivery time of the project may be delayed, which may cause late delivery fees or to lose the benefit of early launching of a new product to a market which can be classified as tardiness cost. To consider these uncertainties, the RCMPSP problem should be considered in a dynamic environment. However, the vast majority of the literature focused on the static side of the problem and literature on the dynamic side of the problem is very limited. The motivation of this thesis is to reduce the impact of these uncertainties by considering a more comprehensive project scheduling model, to suggest appropriate solution approaches for the dynamic environments and to extend the literature of the dynamic extensions of the RCMPSP. For this purpose:

In Chapter 4, we studied the dynamic RCMPSP with uncertain project arrivals. We modelled the problem as an infinite-horizon discrete-time MDP. New project arrivals happen during the time unit. We used DP value iteration to maximise the long-term average profit per unit time. We tested the limits of the DP on the dynamic RCMPSP and generated four test problems. We used two heuristic reactive baseline scheduling methods and a worst-decision DP on the same problems and compared their results with exact results of the DP. We used GA and RBA as heuristic reactive baseline scheduling methods. According to our findings, GA produced closer to optimal results than the simpler heuristic RBA. Since reactive baseline scheduling does not consider the random changes before they occurred, the GA's and the RBA's results are closer to optimal at low arrival probabilities and diverge from optimum at the high arrival probabilities.

In Chapter 5, we studied the dynamic and stochastic RCMPSP with uncertain project arrivals and uncertain task duration as an infinite-horizon discrete-time MDP. We compared the DP, WDA, GA and RBA methods from Chapter 4 on the dynamic and stochastic RCMPSPs, which are created by extending the problems from Chapter 4 with uniformly distributed stochastic task durations, and we included the ORBA method to our comparisons. We also enrich our previous comparison of the deterministic cases from Chapter 4 with results of ORBA. Our approach generates the optimum policies for the dynamic and stochastic RCMPSP and contributes to the literature by extending the work of Melchiors et al. (2018) which only considered single-task projects. According to our findings, the GA produced closer to optimal results than the priority rule heuristic for most of the considered problems. The RBA generally produced results between the GA and the WDP. We observed the same phenomenon with the Chapter 4 that reactive baseline methods are closer to optimal at low arrival probabilities and diverge from optimum at high arrival probabilities. In addition, the GA's and the RBA's results are closer to optimal at deterministic task durations than the stochastic task durations. However, a few exceptions have been observed.

In Chapter 6, we modelled the dynamic and stochastic RCMPSP as an infinite-horizon discrete-time MDP where projects have identical arrival probabilities at each transition time, and tasks have uniformly distributed completion probabilities. The objective of the problem is maximising the expected total discounted long-run profit. We used a linear approximation model to design a practical scheduling policy and showed that it performs near-optimally in small problems and compares favourably to existing heuristics in large problems. Our results show that DP produces statistically significantly better results than ADP for small size problems. However, it suffers from the curse of dimensionality and is not suitable for bigger size problems. ORBA often produced equally high profits as our ADP algorithm in the small size problems. However, ORBA runs in factorial time, and it can not be used in bigger size problems. ADP is the best solution method for the bigger size problems in our comparisons. ADP produced statistically significantly higher profits than RBA and GA in the majority of problems, and GA or RBA outperforms ADP only in a fraction of test problems.

7.2 Managerial insights

This thesis provides an efficient ADP algorithm for dynamic and stochastic RCMPSP that generates statistically significantly higher or equal profits in 88% our comparisons to alternative methods (GA and RBA). Also, this study gives insights

to project managers to determine more suitable methods for their environment by providing a performance comparison of ADP, DP, ORBA, GA and RBA methods in various project settings and various arrival probabilities. We have seen that DP suffers from the curse of dimensionality even for the small size problems, and reactive baseline scheduling methods do not produce close to optimum results at the high arrival probabilities or stochastic task durations. We suggest using DP for problems that are smaller than the computational limitations of DP, such as four projects and two tasks and the reactive baseline scheduling methods such as GA for the environments with very low uncertainties such as 1% new project arrival rates. We don't recommend using GA or other reactive baseline scheduling methods in highly uncertain environments since our test results showed that GA might generate up to 67.2% less average profit per unit time compared to optimal in these environments. We suggest using ADP methods for bigger size problems since our test results showed that ADP is usually statistically significantly more profitable than RBA and GA.

7.3 Future directions

7.3.1 Other dynamic and stochastic project scheduling problems

Since most of the literature focuses on the static problem, some research gaps exist in the dynamic variants of RCMPSP. However, no research considers stochastic resource availability with or without stochastic task duration in the dynamic environment available at the moment. Similarly, other extensions of RCPSP, such as multi-mode problems, can be applied to dynamic environments. Future work might consider other elements of the dynamic project scheduling environment, such as stochastic resource availability, multiple modes of task processing or probability of task processing failure.

The stochastic resource availability considers resource failures or performance failure of human and technical resources, which reduces the available amount of the relating resource type for a period that represents repair time of the resource. The failure of resources may occur in any system. Thus, stochastic resource availability is an important feature to consider in a model to represent real-life more realistically. Stochastic resource availability is a known feature of static RCPSP literature. For example, stochastic RCPSP and stochastic RCMPSP problems which consider stochastic resource availability with or without stochastic task duration are getting attention since 2015 (e.g. Wang et al. (2015); Song et al. (2018); Chand et al. (2019); Zarghami et al. (2019)). Therefore, considering

the stochastic resource availability in dynamic and stochastic RCMPSP may lead to more comprehensive models representing real-life cases. However, the effects of stochastic resource availability can partially be represented just by using the stochastic task durations. Since stochastic resource availability affects task processing, the disruption effect of resource failure can be included in task duration distributions. So rather than considering another stochastic feature, a more comprehensive task duration distributional might be used. The stochastic resource availability can be considered in dynamic and stochastic RCMPSP with minor changes in Monte Carlo simulation of Chapter 6. An ADP method similar to Chapter 6 might be used for this problem if only amounts of free-resources are stochastic. Otherwise, the failure of an allocated resource should disturb the processing of an ongoing task. In that case, pre-emptive task processing or failing and repeating the task processing or increasing the task processing duration with fewer resources should be considered. Thus, an ADP method that has a feature for these uncertainties may generate better results than our model from Chapter 6.

The multiple modes of task processing represent the availability of multiple task processing time options with different resource usage amounts. In other words, task processing time can be shorter with higher resource usages by time unit, or it can be longer with fewer resources usages by time unit. In real life, most task processing duration can be shortened or extended by changing the allocated resource amounts by unit time. This feature is used to process important projects faster or to be able to complete some projects before their due date. The multiple modes of task processing is a well-known feature for the static RCPSP literature. Including the multiple modes of task processing to dynamic and stochastic RCMPSP requires adding the task procession mode selection decision in addition to project acceptance decision and task processing decision. Because of this, the state space and the solution space of the problem will increase enormously, and solution approaches may diverge from the optimum. TThe simplest modelling of this problem would be creating many combinations of available actions with different modes. That would increase the size of the action set enormously, but the ADP method from Chapter 6 can solve that problem model. Our ADP method will be able to differentiate modes by their resource usage amounts. However, the ADP method from Chapter 6 will not be efficient since it tries each element of action set using brute force; thus, the computation time will be rocketed. However, an ADP method that handles the modes selection before creating the action set may be more efficient and generate better results.

There are also other types of problems, for example, failure option of tasks which may lead to project failure or re-processing of the tasks. The task failure

which leads to project failure is considered in dynamic and stochastic RCMPSP by Choi et al. (2007). We could not found any research for the task failure, which leads to re-processing of the task in the dynamic environment. The task failure option is suitable for general PSP, but it is a feature for some PSP applications such as medical trial problems, R&D or new product development. The task failure option can be considered in dynamic and stochastic RCMPSP with minor changes in Monte Carlo simulation of Chapter 6 and our ADP model can be used without any changes to solve this problem.

7.3.2 Other solution approaches

Some of solution approaches for the dynamic RCMPSP and the dynamic and stochastic RCMPSP are explained and used in comparisons in this thesis. Also, the other approaches for these problems in the literature are explained in Chapter 2. Most of the solution approaches for the static RCMPSP can be applied to dynamic problems using the reactive baseline method. However, the ORBA method we used in our comparison represents the best results of the reactive baseline method. Thus we will exclude the static RCMPSP methods in this subsection, and we will focus on other possible methods that approximate the DP. There are three main approximation strategies according to Powell (2021), which are lookup tables, parametric models and nonparametric models.

Lookup tables usually store the information about observed or approximated future rewards for a state and action pair. These values are used during online decision making at the approximate value function. Lookup tables are usually generated by exploring the state space with simulations. Since lookup tables require storing information, they are still prone to the curse of dimensionality for bigger size problems. Choi et al. (2007) used a Q-learning based ADP for the dynamic and stochastic RCMPSP, and they used a lookup table to store their Q-values. Due to the limitation of a lookup table, their model only considers serial project networks, single resource type, single resource usage per task and no project due dates. Powell (2021) states that the approximation strategies with lookup tables are easy to apply and easy to understand, but they can easily become computationally intractable with bigger size problems.

Parametric models approximate a function using a given analytical model via training their coefficients. Parametric models are categorized into linear models and nonlinear models (Powell, 2021). We used a linear model in the Chapter 6. Usually, nonlinear models are used for problems where linear models do not represent them. The nonlinear models are more diverse and can be complicated than the linear models. Some simpler neural networks fit the nonlinear model

category (Powell, 2021). The nonlinear models may represent the problem better than the linear model, but it is harder to find the best-fitted model, and they may require more data to be trained. In Chapter 6, actions of optimal policy and actions of ADP and ORBA are compared. According to this comparison, it is seen that the due date state is a factor for the optimal policy of dynamic and stochastic RCMPSP. Thus, we attempted to create a linear policy that included the remaining number of periods until project is due as a variable, but it did not fit the problem and did not provide profit. Thus, we believe that a nonlinear model that considers the due date state might fit the problem.

Nonparametric models can appropriate any function with arbitrary accuracy at the expense of requiring huge datasets (Powell, 2021). Kernel regression models and Deep natural networks, which both require so much data, are given as examples for the nonparametric models by Powell (2021). He also states that nonparametric models fit the data very closely, and because of this, the noise in the data may affect the results of these methods. Unfortunately, we have not found any paper that uses a nonparametric model for the dynamic RCMPSP or the dynamic and stochastic RCMPSP. Research that uses the nonparametric model for these problems will be the first contributor to the literature.

References

Adhau, S., Mittal, M., and Mittal, A. (2013). A multi-agent system for decentralized multi-project scheduling with resource transfers. *International Journal of Production Economics*, 146(2):646–661.

Adhau, S., Mittal, M. L., and Mittal, A. (2012). A multi-agent system for distributed multi-project scheduling: An auction-based negotiation approach. *Engineering Applications of Artificial Intelligence*, 25(8):1738–1751.

Adler, P. S., Mandelbaum, A., Nguyen, V., and Schwerer, E. (1995). From project to process management: An empirically-based framework for analyzing product development time. *Management Science*, 41(3):458–484.

Ahuja, V. and Birge, J. R. (2020). An approximation approach for response adaptive clinical trial design. *INFORMS Journal on Computing*, 32(4):877–894.

Alhumrani, S. and Qureshi, M. R. (2016). Novel approach to solve resource constrained project scheduling problem (rcpsp). *International Journal of Modern Education and Computer Science (IJMECS)*, 8:60–68.

APM (2012). *APM Body of Knowledge*. Association for Project Management, 6th edition.

Browning, T. R. and Yassine, A. A. (2010). Resource-constrained multi-project scheduling: Priority rule performance revisited. *International Journal of Production Economics*, 126(2):212–228.

Bruni, M. E., Pugliese, L. D. P., Beraldi, P., and Guerriero., F. (2018). A two-stage stochastic programming model for the resource constrained project scheduling problem under uncertainty. In *Proceedings of the 7th International Conference on Operations Research and Enterprise Systems (ICORES)*, volume 1, pages 194–200. INSTICC, SciTePress.

Capa, C. and Ulusoy, G. (2015). Proactive project scheduling in an R&D department a bi-objective genetic algorithm. In *2015 International Conference on Industrial Engineering and Operations Management (IEOM)*, volume 1, pages 1–6.

Chand, S., Singh, H., and Ray, T. (2019). Evolving heuristics for the resource constrained project scheduling problem with dynamic resource disruptions. *Swarm and Evolutionary Computation*, 44:897–912.

Chen, H., Ding, G., Zhang, J., and Qin, S. (2019). Research on priority rules for the stochastic resource constrained multi-project scheduling problem with new project arrival. *Computers & Industrial Engineering*, 137:106060.

Choi, J., Realff, M. J., and Lee, J. H. (2004). Dynamic programming in a heuristically confined state space: A stochastic resource-constrained project scheduling application. *Computers & Chemical Engineering*, 28(6):1039–1058.

Choi, J., Realff, M. J., and Lee, J. H. (2007). A Q-learning-based method applied to stochastic resource constrained project scheduling with new project arrivals. *International Journal of Robust and Nonlinear Control*, 17(13):1214–1231.

Cohen, I., Golany, B., and Shtub, A. (2005). Managing stochastic, finite capacity, multi-project systems through the cross-entropy methodology. *Annals of Operations Research*, 134(1):183–199.

Creemers, S. (2015). Minimizing the expected makespan of a project with stochastic activity durations under resource constraints. *Journal of Scheduling*, 18(3):263–273.

Creemers, S., Leus, R., and Lambrecht, M. (2010). Scheduling markovian pert networks to maximize the net present value. *Operations Research Letters*, 38:51–56.

Davis, M. T., Robbins, M. J., and Lunday, B. J. (2017). Approximate dynamic programming for missile defense interceptor fire control. *European Journal of Operational Research*, 259(3):873–886.

Deblaere, F., Demeulemeester, E., and Herroelen, W. (2011). Proactive policies for the stochastic resource-constrained project scheduling problem. *European Journal of Operational Research*, 214(2):308–316.

Fliedner, T., Gutjahr, W., Kolisch, R., and Melchiors, P. (2012). Solving the dynamic stochastic resource-constrained multi-project scheduling problem with SRCPSP-methods. In *Proceedings of the 13th International Conference on Project Management and Scheduling, Leuven, Belguium: KU Leuven*, pages 148–151.

Frenk, J. B. G. and Kan, A. H. G. R. (1987). The asymptotic optimality of the lpt rule. *Mathematics of Operations Research*, 12(2):241–254.

Gonçalves, J. F., Mendes, J. J., and Resende, M. G. (2008). A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research*, 189(3):1171–1190.

Grey, J. R. (2007). *Buffer Techniques for Stochastic Resource Constrained Project Scheduling with Stochastic Task Insertions Problems*. PhD thesis, University of Central Florida.

Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics (NRL)*, 45(7):733–750.

Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics (NRL)*, 49(5):433–448.

He, F., Yang, J., and Li, M. (2018). Vehicle scheduling under stochastic trip times: An approximate dynamic programming approach. *Transportation Research Part C: Emerging Technologies*, 96:144–159.

Herbots, J., Herroelen, W., and Leus, R. (2007). Dynamic order acceptance and capacity planning on a single bottleneck resource. *Naval Research Logistics (NRL)*, 54(8):874–889.

Holland, J. H. (1992). *Adaptation and Artificial Systems: An Introductory Analysis with Application to Biology, Control, and Artificial Intelligence*. MIT Press, Cambridge, MA, USA.

Homberger, J. (2007). A multi-agent system for the decentralized resource-constrained multi-project scheduling problem. *International Transactions in Operational Research*, 14(6):565–589.

Homberger, J. (2012). A (μ, λ)-coordination mechanism for agent-based multi-project scheduling. *OR spectrum*, 34(1):107–132.

Karam, A. and Lazarova-Molnar, S. (2013). Recent trends in solving the deterministic resource constrained project scheduling problem. In *9th International Conference on Innovations in Information Technology (IIT), Abu Dhabi*, pages 124–129. IEEE.

Khamooshi, H. (1999). Dynamic priority–dynamic programming scheduling method (dp) 2sm: A dynamic approach to resource constraint project scheduling. *International Journal of Project Management*, 17(6):383–391.

Kolisch, R. and Drexl, A. (1996). Adaptive search for solving hard project scheduling problems. *Naval Research Logistics (NRL)*, 43(1):23–40.

Kolisch, R. and Hartmann, S. (1999). *Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis*, pages 147–178. Springer US.

Kolisch, R. and Padman, R. (2001). An integrated survey of deterministic project scheduling. *Omega*, 29(3):249–272.

Kolisch, R. and Sprecher, A. (1997). PSPLIB a project scheduling problem library: Or software-orsep operations research software exchange program. *European Journal of Operational Research*, 96(1):205–216.

Kolisch, R., Sprecher, A., and Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, 41(10):1693–1703.

Li, H. and Womer, N. K. (2015). Solving stochastic resource-constrained project scheduling problems by closed-loop approximate dynamic programming. *European Journal of Operational Research*, 246(1):20–33.

Li, H., Zhang, X., Sun, J., and Dong, X. (2020). Dynamic resource levelling in projects under uncertainty. *International Journal of Production Research*. Advance online publication.

Marinescu, D. C. (2018). Chapter 12 - big data, data streaming, and the mobile cloud. In Marinescu, D. C., editor, *Cloud Computing (Second Edition)*, pages 439–487. Morgan Kaufmann, second edition edition.

Melchiors, P. (2015). *Dynamic and Stochastic Multi-Project Planning*. Lecture Notes in Economics and Mathematical Systems. Springer, Cham, Switzerland.

Melchiors, P. and Kolisch, R. (2009). Scheduling of multiple R&D projects in a dynamic and stochastic environment. In *Operations Research Proceedings 2008*, pages 135–140. Springer, Heidelberg.

Melchiors, P., Leus, R., Creemers, S., and Kolisch, R. (2018). Dynamic order acceptance and capacity planning in a stochastic multi-project environment with a bottleneck resource. *International Journal of Production Research*, 56(1-2):459–475.

Mori, M. and Tseng, C. C. (1997). A genetic algorithm for multi-mode resource constrained project scheduling problem. *European Journal of Operational Research*, 100(1):134–141.

Ortiz-Pimiento, N. R. and Diaz-Serna, F. J. (2018). The project scheduling problem with non-deterministic activities duration: A literature review. *Journal of Industrial Engineering and Management (JIEM)*, 11(1):116–134.

Pamay, M. B., Bülbül, K., and Ulusoy, G. (2014). *Dynamic Resource Constrained Multi-Project Scheduling Problem with Weighted Earliness/Tardiness Costs*, volume 200 of *International Series in Operations Research & Management Science*, pages 219–247. Springer US.

Parizi, M. S., Gocgun, Y., and Ghate, A. (2017). Approximate policy iteration for dynamic resource-constrained project scheduling. *Operations Research Letters*, 45(5):442–447.

Patterson, J. H. (1984). A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. *Management science*, 30(7):854–867.

Powell, W. B. (2009). What you should know about approximate dynamic programming. *Naval Research Logistics*, 56(3):239–249.

Powell, W. B. (2011). *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, volume 842 of *Wiley series in probability and statistics*. Wiley.

Powell, W. B. (2021). *Reinforcement learning and stochastic optimization*. Wiley, Hoboken, New Jersey.

Ronconi, D. P. and Powell, W. B. (2010). Minimizing total tardiness in a stochastic single machine scheduling problem using approximate dynamic programming. *Journal of Scheduling*, 13:597–607.

Rostami, S., Creemers, S., and Leus, R. (2018). New strategies for stochastic resource-constrained project scheduling. *Journal of Scheduling*, 21(3):349–365.

Sammut, C. and Webb, G. (2010). *Decision Epoch*. Springer US.

Satıç, U. (2014). Çok kaynak kısıtlı projelerin sezgisel yöntemlerle çizelgelenmesi. Yildiz Technical University.

Satic, U., Jacko, P., and Kirkbride, C. (2020a). Performance evaluation of scheduling policies for the DRCMPSP. In Gribaudo, M., Sopin, E., and Kochetkova, I., editors, *Analytical and Stochastic Modelling Techniques and Applications*, volume 12023, pages 100–114, Cham. Springer International Publishing.

Satic, U., Jacko, P., and Kirkbride, C. (2020b). Performance evaluation of scheduling policies for the dynamic and stochastic resource-constrained multi-project scheduling problem. *International Journal of Production Research*. Advance online publication.

Schütz, H.-J. and Kolisch, R. (2012). Approximate dynamic programming for capacity allocation in the service industry. *European Journal of Operational Research*, 218(1):239–250.

Schwindt, C. (1998). Generation of resource constrained project scheduling problems subject to temporal constraints. Report WIOR-543, Universitat Karlsruhe, Kaiserstrasse 12, D-76128 Karsruhe, Germany.

Song, W., Xi, H., Kang, D., and Zhang, J. (2018). An agent-based simulation system for multi-project scheduling under uncertainty. *Simulation Modelling Practice and Theory*, 86:187–203.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, second edition. edition.

Tereso, A. P., Araújo, M. M. T., and Elmaghraby, S. E. (2004). Adaptive resource allocation in multimodal activity networks. *International Journal of Production Economics*, 92(1):1–10.

Tijms, H. C. (1994). *Stochastic Models: An Algorithmic Approach*. John Wiley & Sons.

Ulusoy, G. (2002). Proje planlamada kaynak kısıtlı çizelgeleme. *Yöneylem Araştırması: Halim Doğrusöz'e Armağan*, pages 89 – 128.

Wang, X., Chen, Q., Mao, N., Chen, X., and Li, Z. (2015). Proactive approach for stochastic rcmpsp based on multi-priority rule combinations. *International Journal of Production Research*, 53(4):1098–1110.

Wang, Y., He, Z., Kerkhove, L.-P., and Vanhoucke, M. (2017). On the performance of priority rules for the stochastic resource constrained multi-project scheduling problem. *Computers & Industrial Engineering*, 114:223–234.

Wellingtone PPM (2018). The state of project management annual survey 2018. http://www.wellingtone.co.uk/wp-content/uploads/2018/05/The-State-of-Project-Management-Survey-2018-FINAL.pdf. Accessed July 7, 2021.

Yassine, A. A., Mostafa, O., and Browning, T. R. (2017). Scheduling multiple, resource-constrained, iterative, product development projects with genetic algorithms. *Computers & Industrial Engineering*, 107:39–56.

Zarghami, S. A., Gunawan, I., de Zubielqui, G. C., and Baroudi, B. (2019). Incorporation of resource reliability into critical chain project management buffer sizing. *International Journal of Production Research*, 0(0):1–15.