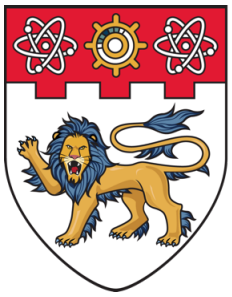


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

PROJECT SCHEDULING IN DISTRIBUTED AND DYNAMIC ENVIRONMENTS

WEN SONG
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
2018

Project Scheduling in Distributed and Dynamic Environments

Wen Song

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

2018

Abstract

Project scheduling is an important task of modern business management. Classic project scheduling approaches assume a centralized and deterministic environment. However, today's manufacturing and management have entered into a more open and dynamic environment, which jeopardizes the effectiveness of traditional approaches. Two important practical factors that cause this issue are: 1) *distributed management*, where multiple decision makers with conflicting individual objectives are involved in the scheduling process, and 2) *execution uncertainty*, where projects are executed in dynamic environments containing various uncertainty sources. It is non-trivial to incorporate these practical factors since they make the project scheduling problems, which are already computationally intractable, even harder to solve. In this thesis, we provide effective approaches to address the two above-mentioned practical factors.

We first study the scheduling problem in a distributed multi-project setting, where each project is controlled by an autonomous project agent. Classic centralized approaches cannot be applied in such a distributed multi-agent environment. However, existing distributed approaches encounter difficulties in dealing with large problems while preserving information privacy of project agents. We design a novel distributed approach based on multi-unit combinatorial auction, which does not require sensitive project information. To handle the hard valuation problem of the participators, we introduce the capacity query to efficiently elicit useful information from the project agents. We then design two allocation strategies that work with the capacity query to find good schedules, including a greedy strategy and a branch-and-bound heuristic. Empirical results indicate that the two strategies can find good solutions with higher quality than state-of-the-art distributed approaches, and scale well to large problem instances.

We next study the risk-neutral proactive scheduling problem with uncertain activity durations. More specifically, we aim at finding an optimal project execution strategy that minimizes the expected makespan. Traditional approaches assume that the uncertain duration of an activity can be modeled as a random variable that does not depend on its start time. However, this can be violated in many real-world scenarios. In this work, we generalize the traditional time-independent model to support the time-dependent workability uncertainty, which has not been stud-

ied before and does make the activity duration time-dependent. Since the resultant discrete stochastic optimization problem is hard to solve, we propose a principled approximate approach based on Sample Average Approximation (SAA). By exploiting interesting problem properties, we design two efficient branch-and-bound algorithms to optimally solve the SAA problem. The effectiveness of our approach is verified by the experiments on multiple uncertainty models, including a real-world workability uncertainty distribution.

Finally, we study the risk-aware proactive scheduling problem, which tries to optimize the robust makespan instead of expected makespan. Robust makespan is considered to be more practical in real-world applications, since it constraints the actual makespan within certain (probabilistic) risk level. State-of-the-art approaches for this problem are based on probabilistic constrained optimization, which leads to complex Mixed Integer Linear Programs that must be heuristically approximated. Instead, we propose a principled approximate approach by optimizing the robust makespan via Conditional Value-at-Risk (CVaR). However, existing CVaR optimization methods assume linear solution spaces, and hence are not applicable to our problem due to the combinatorial nature of resource-constrained scheduling. Hence, we design a general branch-and-bound framework for CVaR optimization in combinatorial spaces. We then instantiate this framework by adapting the branch-and-bound algorithms designed in our previous work to solve the risk-aware proactive problem. Results confirm that our approach scales well to a large number of samples, and can produce much better solutions than state-of-the-art approaches.

To sum up, we have proposed a series of approaches to cope with challenging project scheduling problems with practical factors including distributed management and execution uncertainty. These contributions can also shed light on solving more complex and practical scheduling problems.

Acknowledgment

First of all, I would like to thank my advisor, Prof. Jie Zhang, who has been continuously kind, patient and supportive to me. I thank him for introducing me to the exciting field of Artificial Intelligence, motivating me to perfect my research ideas, and teaching me how to become a qualified researcher. I cannot succeed in pursuing my doctoral degree without his guidance. I also want to thank my collaborators, Dr. Donghun Kang and Dr. Hui Xi, for all their valuable efforts in discussing research, sharing experience, and revising papers.

During the last four years, I have been fortunate to work in a wonderful research lab. I would like to thank my colleagues and friends in Rolls-Royce@NTU Corp Lab and School of Computer Science and Engineering. They include Zehong Hu, Pengfei Wei, Sa Gao, Yukun Ma, Guanghao Zhang, Zhiguang Cao, Zhu Sun, Dongxia Wang, Zhenchao Bing, Quanchi Weng, and Shuo Chen.

Outside the academic world, I sincerely thank my friends Xiaodong Song, Hao Cong, Mao Mao, and Lu Ren. They are always available, whenever I need their help or want to share my joy. Without their support, I cannot survive in the long journey of Ph.D. study.

Finally, I give the deepest gratitude to my parents. Their endless love and unconditional support to me is simply beyond words.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Most modern business corporations need to deal with various scheduling problems on a daily basis, to arrange, control and optimize their business processes (Schwindt et al., 2015). On one hand, the quality of schedules has direct impact on the competitiveness of the business organizations on the market (Kang et al., 2017). On the other hand, however, scheduling problems (e.g. job shop scheduling, project scheduling) are generally hard combinatorial optimization problems that are computationally intractable. Motivated by the scientific and business values, scheduling problems have been widely studied by researchers from the communities of both Operations Research and Artificial Intelligence in the past few decades, and a variety of techniques (e.g. mathematical programming, constraint optimization, heuristic methods) have been successfully developed.

In this thesis, we focus on the Resource-Constrained Project Scheduling Problem (RCPSP), which is a very general model that can be applied to many business applications, such as manufacturing process, software development, workflow scheduling for cloud computing (Brucker and Knust, 2012). This problem concerns with determining the start times of a set of resource-requiring activities to optimize certain performance criterion (e.g. makespan), such that the resource capacities and temporal constraints between activities are respected. RCPSP is known to be NP-hard (Blazewicz et al., 1983).

Traditionally, RCPSP is often assumed to be a centralized and deterministic problem, where only one decision maker exists and all information is known in advance. However, with the rapid growth of the scale and complexity, business processes in today's organizations have become more open and dynamic which in-

validate classic approaches developed under those idealized assumptions. Two major practical factors that need to be considered in real-world applications are:

- **Distributed management**. It is quite common for business firms to conduct intra- and inter-firm collaborations, for the purpose of improving efficiency and reducing cost (Walsh and Wellman, 2003). This trend inevitably brings the traditional centralized scheduling into a distributed environment, involving multiple decision makers with different individual objectives. In addition, these decision makers may not be willing to share all their private information, since they may be competitors in the same marketplace. Traditional centralized approaches are not capable of dealing with these new features. Hence, distributed approaches that consider both distributed decision making and information privacy are needed.

- **Execution uncertainty**. The execution of activities in real-world projects is often sensitive to various sources of uncertainty (e.g. transportation time, manpower availability, weather changes). Schedules obtained by solving deterministic problems are much less useful under execution uncertainty, mainly for two reasons. Firstly, optimal solutions obtained by optimizing the deterministic performance criterion (e.g. makespan) can hardly be optimal in the actual execution. Secondly, feasible deterministic schedules could quickly become infeasible during actual execution. Hence, it is very important to consider the possible uncertainties in modeling and solving the scheduling problems.

Since RCPSP itself is already difficult to solve, incorporation of the above mentioned practical factors is very challenging, both in modeling the problems (e.g. how to represent decision makers' preferences, how to model different uncertainty sources) and designing efficient algorithms (e.g. how to address the complexity in preference computation, how to tackle the stochasticity). In this thesis, we address these practical problems by contributing novel algorithms that are carefully designed by exploiting important and interesting properties of the scheduling problems. Specifically, for distributed management, we study the problem of distributed multi-project scheduling. For execution uncertainty, we study two problems including risk-neutral and risk-aware proactive scheduling under uncertain activity

durations. In the following sections, we give detail introduction on these problems and our contributions.

1.1 Distributed Multi-Project Scheduling

Multi-project scheduling is common in project management (Xi et al., 2015). When all projects are centrally controlled by one decision maker, the (resource-constrained) multi-project scheduling problem can be readily reduced to RCPSP by leveling the multi-project structure to create a "super" project with all activities being incorporated (Hartmann and Briskorn, 2010). Then, algorithms for RCPSP can be applied on the super project to solve the multi-project scheduling problem. However, when the assumption of centralized decision making does not hold, the reduction to RCPSP is not an ideal solution since it requires full control and complete information of all projects. In practice, distributed management in multi-project scheduling is not rare. This problem, formally termed as the *Distributed Resource-Constrained Multi-Project Scheduling Problem* (DRCMPSP) (Confessore et al., 2007), has received much attention recently. In DRCMPSP, projects are controlled by different autonomous decision makers. To achieve individual objectives, these decision makers usually need to compete for some shared global resources with limited capacities.

Many real-world problems in manufacturing and service operations with complex product/supply structures have been considered as DRCMPSP. In (Kutanoglu and Wu, 1999), an intra-firm scenario is given where different product managers in an electronics manufacturing company must share and compete for shared production resources (e.g. automated production lines) on a regular basis. To satisfy their own customers' requirements, these managers need to deal with uniquely different sets of activities and constraints (e.g. lead time constraint, customer-specific processing requirements). Regarding inter-firm scenarios, a typical example can be found in the Aero Repair and Overhaul industry (Stranjak et al., 2008). In this example, different fleet managers dedicated to managing the aero engines of an airline, compete with other fleet managers for several overhaul bases that have limited repair capacities. These managers need to schedule the overhaul base visit time for each controlled engine, to maximize the operating revenue of each fleet. Similar inter-firm cases can also be found in the airport ground handling service scheduling (Mao et al., 2009)

and supply chain scheduling (Lau et al., 2006). In a distributed environment, the approach that respects the privacy of self-interested decision makers is preferable as stated in (Lau et al., 2006; Stranjak et al., 2008; Vytelingum et al., 2009), because decision makers may be potential competitors in the same marketplace.

Due to its distributed nature, DRCMPSP is usually modeled as a mediated multi-agent system, where each project is represented by a Project Agent (PA) and all PAs are coordinated by a Mediator Agent (MA). The core problem in DRCMPSP is how to allocate the shared global resources to each PA without private project information. (Multi-unit) Combinatorial auction is an ideal choice for modeling this kind of distributed resource allocation problems. Naturally, when modeling DRCMPSP as an auction, the PAs are the bidders which compete for scarce resources controlled by the MA, which is an auctioneer. In such an auction model of DRCMPSP, the auctioneer requires only high-level valuation from the bidders on the scarce shared resources to make allocation decisions, which can satisfy the privacy requirement of the decision makers (Wellman et al., 2001). Also, it allows the bidders to express complex preferences on combinations of items, which are called *bundles* in single-unit scenarios and *multisets* in multi-unit scenarios.

To date, however, existing combinatorial auction based approaches for DRCMPSP suffer from several drawbacks (will be detailed in Section 2.2). Firstly, the DRCMPSP models studied are rather simple (e.g. single-unit resources, small activity sets). Secondly, though the PAs are allowed to bid for bundles of items, there is no formal formulation of the DRCMPSP as a combinatorial auction (except (Wellman et al., 2001) which studies a much simpler problem than ours). Thirdly, these approaches underperform both in solution quality and computational efficiency, and cannot scale to large practical problem cases with thousands of activities from tens of projects.

Against this background, in (Song et al., 2017a), we design a novel combinatorial auction based approach for solving complex DRCMPSP with multi-unit resources and complex activity precedence constraints (preliminary conference version is (Song et al., 2016)). Following previous research on this topic, we assume the bidders are willing to reveal their true valuations on different resource allocation decisions, i.e. currently we do not handle the incentive compatibility issue.[1] We first formulate

[1]Note that privacy and truthfulness are two separate issues. Even though an agent is willing

DRCMPSP as a multi-unit combinatorial auction problem, and formally show that the social welfare maximizing allocation of the resulting auction minimizes the total delay cost of the DRCMPSP, when the revenue of each project is sufficiently high. Then, to resolve the hard problem for the bidders in generating preferences (i.e. evaluation), we introduce the *capacity query*, which elicits the (approximate) valuation of a bidder by asking it to solve a bidding problem with certain item capacity profile. We show that the bidding problem can be solved by solving a RCPSP with time-varying resource capacities. Finally, we adopt two strategies introduced in (Gonen and Lehmann, 2000) that work with capacity queries to find good allocations for the DRCMPSP auctions.

The first strategy maintains a series of query asking and greedy resource allocation processes, where one bidder will be granted the required resources after each round of query asking and will not be asked again. The granted bidder is chosen according to the average price criterion (Gonen and Lehmann, 2000). We show that when the fixed unit pricing scheme is used with the greedy allocation strategy, a bidder will only bid for a special type of multisets which represents compact resource utilization profiles of its project. We further show that when the bidding problems can be solved optimally, the worst-case approximation guarantee for the winner determination problem proved in (Gonen and Lehmann, 2000) still holds in the DRCMPSP auctions (under reasonable assumption that the total required resource units by each bidder is sufficiently large).

The greedy allocation strategy can rapidly find a feasible allocation for the DRCMPSP auction, where all bidders are allocated in a specific sequence decided by the average price. However, some other sequences that can result in better social welfare may be ignored. The second strategy employs a branch-and-bound process which aims at finding a better sequence of allocating the bidders (Gonen and Lehmann, 2000). This strategy can effectively improve the results of the greedy allocation, though it requires higher computational effort for the bidders. In addition, we employ the VCG-based payment scheme proposed in (Nisan and Ronen, 2007) to guarantee that backtracking will not decrease bidders' utilities.

---

to convey its valuations truthfully, it may still not be willing to disclose sensitive information (e.g. activity durations and resource requirements, local resource capacities) (Fink and Homberger, 2015).

We conduct extensive experiments to evaluate the performance of our approach on benchmarking DRCMPSP instances with multi-unit resources (obtained from the public benchmark MPSPLIB) and single-unit resource (generated from MPSPLIB). Results show that the two strategies can generate solutions with higher quality than state-of-the-art distributed approaches, and can scale to large problems with reasonable computational time.

1.2 Risk-neutral Proactive Scheduling

Many of the real-world applications of RCPSP involve considerable amounts of uncertainty sources. For example, activities may take more or less time to finish than expected, material transportation may be delayed, resource availability may vary over time, new activities may have to be inserted, etc. In such kind of dynamic situation, the schedules obtained by solving the deterministic RCPSP instances without considering the uncertainty information is not suitable for executing the projects. Firstly, uncertain durations could disrupt the deterministic schedules quite frequently, which requires intensive efforts to restore the schedule feasibility. Secondly, the actual project schedules can deviate much from the original deterministic ones, leading to significant degradation in the quality of the original schedules. Hence, scheduling models and approaches incorporating uncertainties are of great practical values (Herroelen and Leus, 2005).

In Song et al. (2017b), we address the problem of RCPSP with uncertain activity durations. According to the taxonomy in (Bidot et al., 2009), approaches for scheduling in stochastic environment can be classified into three groups, including (a) proactive approaches which generate baseline solutions that make complete decisions (e.g. start-time schedules) or partial decisions (e.g. flexible policies) before execution, (b) revision approaches which modify the baseline solutions during execution, and (c) progressive approaches which make all decisions in an online fashion and generate no baseline solutions. Compared with the other two alternatives, proactive approaches tend to produce solutions with higher quality and robustness (Bidot et al., 2009). Moreover, decisions made in the baseline solutions (e.g. start times, resource allocation commitments) can provide important support and visibility for better preparing and coordinating the actual execution (Lamas

and Demeulemeester, 2016).

Till now, a number of proactive approaches for RCPSP with stochastic durations have been successfully developed in the literature. Most of these approaches are designed for the model of stochastic RCPSP (Creemers, 2015), where the activity duration is assumed to be a random variable having no relation to its scheduled start time. In reality, however, this assumption could be violated quite often since it cannot reflect all the sources of uncertainty (Bruni et al., 2015). In fact, time-dependent uncertainties exist widely in real-world situations, for instance the daily traffic patterns of the transportation networks, and the seasonality of the weather conditions. For the scheduling problem, we give an motivating example as follows. Consider a quality assurance project with a series of product testing activities, each of which can be executed (i.e. workable) only when certain weather conditions (e.g. temperature, humidity, wind speed, and so on) are satisfied. Furthermore, an activity needs to secure enough workable days to finish successfully. In this case, the activity duration uncertainty comes from the uncertain *workability* of each time slot (day). Taking seasonality into account, the duration uncertainty of an activity should depend on its start time (e.g. July or December), which contradicts the assumption of time-independence in previous research.

To resolve this issue, in (Song et al., 2017b), we relax the time-independent assumption by generalizing the traditional stochastic RCPSP model to incorporate the *time-dependent workability uncertainty* (Song et al., 2017c), which indeed causes the random durations to be time-dependent. We adopt Partial-order Schedules (POS) (Policella et al., 2004) as proactive solutions, which is a type of flexible executing policy for RCPSP. Compared to start-time schedules, POS is more flexible in handling unforeseen events in the execution time (Fu et al., 2012). Based on the generalized uncertainty model, we first study a risk-neutral problem, i.e. finding an optimal POS that minimizes the *expected makespan*. We formulate the proactive scheduling problem as a discrete stochastic optimization problem. However, it is not trivial to find the optimal POS, not only due to the intractability of RCPSP. Accounting for stochastic duration is more challenging, since even evaluating a given solution is intractable. It has been shown that for the basic stochastic RCPSP model without resource constraints and time-dependent workability uncertainty, it is #P-complete to compute the expected makespan (Hagstrom, 1988).

In (Song et al., 2017b), we propose to approximate the proactive scheduling problem based on Sample Average Approximation (SAA) (Kleywegt et al., 2002), which is a principled approximation scheme for solving hard discrete stochastic optimization problems, with the proven ability to converge to the optimal solution. Our approach first generates a set of samples from the probability distributions that describing the uncertain durations, then optimally solves the resulting SAA problem. As we will show later, the SAA problem is NP-hard due to the combinatorial nature of RCPSP. Therefore, we design two branch-and-bound algorithms that search for the optimal solution of the SAA problem from two aspects. The first algorithm uses a constructive approach to link the activities one by one to a partial solution using feasible resource flows. The second algorithm iteratively detects and resolves resource conflicts between activities by adding precedence constraints. We empirically compare the solutions produced by our approaches and state-of-the-art POS generation approaches on uncertainty distributions from existing works and real-world dataset, and the results confirm that our approach can generate high-quality solutions.

1.3 Risk-aware Proactive Scheduling

For proactive scheduling problems, though expected makespan is the most intuitive and commonly adopted criterion for evaluating different proactive solutions, it cannot reflect the *risk* of the outcome obtained from executing a proactive solution. This is a serious limitation when risk is of great importance to the decision maker (e.g. managing projects for capital-intensive industries), since there could be a high chance that the actual makespan is much larger than the expected value (Beck and Wilson, 2007). In the literature, several works (Beck and Wilson, 2007; Fu et al., 2012; Varakantham et al., 2016; Fu et al., 2016) propose to conduct risk-aware proactive scheduling by minimizing the *robust makespan*. Essentially, robust makespan focuses on controlling the probability that the actual makespan exceeds a threshold value within a predefined risk parameter $\alpha \in (0, 1)$, therefore this objective is also referred to as α-robust makespan.

Similar to the risk-neutral problem, existing risk-aware approaches often resort to sampling-based techniques to mitigate the complexity brought by the stochastic

activity durations. For example, approaches in (Beck and Wilson, 2007) and (Fu et al., 2012) use sampling and simulation to evaluate solutions. State-of-the-art approaches in (Varakantham et al., 2016; Fu et al., 2016) consider α-robust makespan minimization as a probabilistic constrained problem, which can be approximated by SAA with the guarantee of converging to the optimal solution (Luedtke and Ahmed, 2008). However, these approaches result in complex Mixed Integer Linear Programs (MILP) that are computationally prohibitive even with sophisticated commercial solvers. To scale up the MILPs, (Varakantham et al., 2016; Fu et al., 2016) propose a summarization heuristic to aggregate multiple samples into a representative one, and then solve a deterministic RCPSP built on the representative sample to obtain the proactive solution. However, this heuristic compromises the convergence guarantee, and decreases solution quality. In addition, the MILP formulations and the summarization heuristic is not applicable to the generalized duration model with the time-dependent workability uncertainty.

In (Song et al., 2018), we propose to optimize the α-robust makespan via CVaR, a popular measure in risk-sensitive decision making problems (Rockafellar and Uryasev, 2002). Our approach scales up to hundreds of samples without the need of sample summarization, hence can provide better robust makespan and more precise control of the risk parameter α. Based on the expectation form of CVaR minimization, we approximate the proactive scheduling problem using SAA, which results in an NP-hard combinatorial problem similar to the risk-neutral setting. This also excludes the traditional CVaR minimization approaches that assume the decision space is linear (Hong et al., 2014). Thus, we design a general branch-and-bound framework for CVaR minimization in combinatorial space. Based on this framework, we modify the two algorithms designed for minimizing the expected makespan to solve the risk-neutral problem. These algorithms also support the generalized uncertain duration model, which cannot be solved by current risk-aware proactive scheduling approaches. Our numerical results show that our algorithms scale well to large sample sizes, and can produce solutions with significantly lower α-robust makespan than state-of-the-art approaches.

1.4 Thesis Organization

The rest of this thesis is organized as follows.

- In Chapter 2, we review existing works that are related to the topics studied in the thesis, including approaches for basic RCPSP, DRCMPSP, and proactive scheduling problems.

- In Chapter 3, we present our approach to solve the distributed multi-project scheduling problem. We describe our combinatorial auction formulation of the distributed scheduling problem in great detail, and provide theoretical analysis on the correctness of the formulation. We then present our approaches to solve the combinatorial auction problem, along with experimental results on benchmarks instances.

- In Chapter 4, we describe our work on solving risk-neutral proactive scheduling problems. We first generalize the traditional uncertain duration model to incorporate the time-dependent workability uncertainty, and formulate our proactive problem on this generalized model. We then show how to approximate the proactive problem using SAA, and design two efficient branch-and-bound algorithms to solve the SAA problem optimally. Experimental results on multiple uncertainty models are provided to validate the effectiveness of our algorithms.

- In Chapter 5, we study the proactive scheduling problem in the risk-aware setting. We briefly describe some important concepts regarding risk-aware decision making, and then formulate our CVaR based proactive problem. Next, we present our general framework for combinatorial CVaR optimization, and explain how to instantiate this framework to solve the proactive scheduling problem. Empirical analysis and comparison with benchmarking algorithms are provided to confirm the advantages of our algorithms.

- In Chapter 6, we conclude the thesis by summarizing our contributions, and pointing out promising directions for future studies.

Chapter 2

Literature Review

In this chapter, we review existing works that are related to our research topics. We first give a brief description on approaches for solving the basic RCPSP in Section 2.1. Then in Sections 2.2 and 2.3, we provide detailed reviews of approaches for DRCMPSP and proactive scheduling, respectively.

2.1 Traditional Methods for RCPSP

In the literature, a large volume of works have been done in solving basic RCPSP, which is a centralized and deterministic problem. In this section, we give a brief overview of these approaches. Instead of providing a comprehensive survey, our aim here is to outline typical methodologies for tackling this hard combinatorial optimization problem, which can serve as the basic foundation for solving more sophisticated and practical problems.

In general, all approaches for solving RCPSP can be classified into two groups, i.e. exact approaches that have proven ability to find the optimal solution, and heuristic approaches that trade off the solution optimality with computational efforts. Among the exact approaches, a large number of them focus on developing different Mixed Integer Linear Programming (MILP) models of RCPSP, which can then be solved by standard solvers (e.g. CPLEX and Gurobi). Several types of MILP formulations are available, such as discrete-time formulations (Pritsker et al., 1969), continuous-time formulations (Artigues et al., 2003), and event-based formulations (Koné et al., 2011). However, these MILP models are not scalable and only work on small instances (Brucker and Knust, 2012). On the other hand, another type of

exact approaches are designed based on constraint programming (Rossi et al., 2006), which combines backtracking search and constraint propagation techniques to find the optimal schedule. Datails about typical constraint propagation techniques for RCPSP such as timetabling, edge-finding, energetic reasoning, and lazy clause generation can be found in (Baptiste et al., 2012; Laborie, 2003; Schutt et al., 2013). Compared with MILP based approaches, constraint programming based approaches often exhibit significantly better computation efficiency, due to the active exploration of constraints for reducing search spaces. Though exact approaches require exponential time in the worst case to find the optimal solution, most of them are anytime algorithms which can be terminated early with high-quality solutions.

Considerable amounts of heuristic approaches for RCPSP are also available in the literature. Two most commonly used approaches are schedule generation schemes (SGS) with priority rules, and metaheuristic approaches. The basic idea of SGS is to schedule all activities based on a sequence determined by certain priority rules. Due to the simplicity and reasonable solution quality, SGS has been widely used in designing heuristic algorithms for solving RCPSP (Kolisch, 1996; Browning and Yassine, 2010; de Nijs and Klos, 2014). Metaheuristic is another class of heuristic RCPSP approaches, which employ certain random components in designing the searching algorithms. Typical metaheuristics include Tabu Search, Simulated Annealing, Genetic Algorithm, etc. The detailed review and empirical comparison of different metaheuristics can be found in (Kolisch and Hartmann, 2006; Liao et al., 2011; Van Peteghem and Vanhoucke, 2014). Though heuristic algorithms normally require less computation time than exact algorithms, the absence of any theoretical guarantee on solution optimality is a major drawback.

2.2 Distributed Approaches for DRCMPSP

In this section, we review existing research on DRCMPSP. Since our approach for solving this problem is based on combinatorial auction, we review existing approaches by putting them into two categories, the combinatorial auction based approaches in Section 2.2.1 that are built on the similar basis as ours and other approaches in Section 2.2.2. Our approach also shares the spirit of preference elicitation, which aims at reducing the computational burden for an agent in evaluating

different allocations. Therefore we give a brief review of preference elicitation methods in Section 2.2.3.

2.2.1 Combinatorial Auction based Approaches

In the literature, combinatorial auction has been applied to solve simple distributed scheduling problems (Kutanoglu and Wu, 1999; Wellman et al., 2001; Confessore et al., 2007). In (Kutanoglu and Wu, 1999), the authors introduce a combinatorial auction based distributed scheduling framework to solve the distributed job shop scheduling problem, a special case of DRCMPSP. This approach simulates the Lagrangian decomposition approach of the centralized problem in a distributed manner, by allowing project agents to iteratively bid for some combinations of machine time slots. Nevertheless, this approach cannot guarantee finding feasible schedules for all the projects through the bidding process, hence a centralized algorithm which requires all information is needed as a post-processing step to clear the market and generate a feasible solution.

In (Wellman et al., 2001), the authors study a simple factory scheduling model, where a group of agents compete for the limited time slots in a common factory. Each agent has a single activity needed to be processed, which is associated with a profit value and a hard deadline. A combinatorial auction based protocol is studied, and the authors give concrete theoretical results on the price equilibrium and allocation efficiency. However, their analysis is limited to scenarios where each agent only owns one activity, which is much simpler than the problem we studied. One major simplification is that, the bidding agents in (Wellman et al., 2001) have no difficulty in valuation computation, while in our case it could be intractable for a bidder to compute the exact value on a given multiset, as will be shown in Section 3.2.

In (Confessore et al., 2007), an approach based on an iterative combinatorial auction named *i*Bundle (Parkes and Ungar, 2000) is proposed. *i*Bundle has been applied to solve problems such as multi-agent pathfinding (Amir et al., 2015) and train scheduling (Parkes and Ungar, 2001). The desirable properties of *i*Bundle, including the optimality guarantee and (myopic) strategy-proofness, are based on the assumption that all bidders can give Myopic Best-Response (MBR) to the bundle prices given in each round of auction. However, when applied to DRCMPSP, it is difficult to satisfy MBR due to the intractability of local bidding problems.

When approximate algorithms without optimality guarantee are used for bidding, as in (Confessore et al., 2007), both the optimality of final solutions and strategy-proofness are compromised. In addition, when MBR cannot be satisfied, *i*Bundle cannot guarantee that all bidders will be granted a bid upon termination. Finally, *i*Bundle could take a large number of iterations to terminate, which requires intense computation for bidders and the auctioneer, resulting in inefficiency on large-scale cases.

Except in (Wellman et al., 2001) which studies a much simpler problem than the DRCMPSP model in this chapter, the relations between the solutions of DRCMPSP and combinatorial auction have not been analyzed in the current literature. We fill this gap in Section 3.2 by providing a combinatorial auction formulation of DRCMPSP, along with theoretical analysis on the relations between the allocations of the auction and solutions of DRCMPSP. In addition, all approaches in (Kutanoglu and Wu, 1999; Wellman et al., 2001; Confessore et al., 2007) can only be used to schedule single-unit global resources. And, test cases are rather small with tens of activities in total from several projects. In contrast, our approach can efficiently handle multi-unit resources and large problem cases.

2.2.2 Other Approaches

More recently, several approaches have been proposed to solve much larger DRCMPSP cases with hundreds to thousands of activities from tens of projects sharing several multi-unit global resources (Mao et al., 2009; Homberger, 2012; Adhau et al., 2012; Zheng et al., 2014). In (Mao et al., 2009), the authors propose a market-based approach to schedule the airport ground handling services, but the resource capacities are assumed to be infinite which is hardly found in practice. In (Homberger, 2012), an evolutionary computation based negotiation approach is presented, but it is outperformed in solution quality by a centralized approach SASP (Kurtulus and Davis, 1982), one of the best priority-rule based multi-project scheduling algorithms. In general, priority-rule based approaches solve multi-project scheduling problems in a centralized fashion, hence cannot satisfy the requirement of DRCMPSP. Nevertheless, they are often used as benchmarks for evaluating the performance of DRCMPSP approaches, since some evaluation criteria studied in DRCMPSP, such as average project delay (see Section 3.1), also exist in centralized multi-project scheduling.

In (Adhau et al., 2012), the authors introduce an approach named DMAS/ABN, which conducts an auction-based negotiation on each time slot for each activity. In (Zheng et al., 2014), an approach named DMAS/EM is proposed, which employs an activity elimination algorithm to fix an infeasible solution. Both DMAS/ABN and DMAS/EM can generate better solutions than SASP.

One common feature of the approaches in (Mao et al., 2009; Homberger, 2012; Adhau et al., 2012; Zheng et al., 2014) is that, they are based on activity-level negotiation to generate or fix a solution, which has two major drawbacks. Firstly, activity information (e.g. start time, duration, resource requirements) is inevitably required by the mediator. Note that in a competitive environment, the project agent may still be unwilling to reveal its valuable sensitive information, even if the mediator is an automated agent (Vytelingum et al., 2009). Secondly, when the individual objective of each PA cannot be decomposed precisely to each activity (e.g. project, delay cost), the decisions on global resource allocation to each activity can only rely on the estimated objective value, which could result in unsatisfactory solution quality. On the contrary, global resource allocation in our approach is purely on the project level, which can satisfy the private information requirement and provide more precise information for allocating global resources.

2.2.3 Preference Elicitation

In general, preference elicitation is a type of frameworks and methods that try to make optimal or near-optimal decisions without the need of fully knowing the complete preference profile of each agent. Combinatorial auction is a typical application of preference elicitation. It is well known that the number of bundles grows exponentially with the number of items ($2^k - 1$ bundles for k items) in combinatorial auction. In the worst case, a bidder needs to compute its value on all these bundles, which is unacceptable in practice. Preference elicitation for combinatorial auction tries to address this issue by reducing the number of bundles that need to be evaluated by a bidder (Conen and Sandholm, 2001).

A general elicitation framework for combinatorial auction is introduced in (Hudson and Sandholm, 2004). This framework maintains a set of candidate allocations, and iteratively asks certain type of queries to the bidders to prune the candidate set until it is provably optimal. Conen and Sandholm propose another elicitation frame-

work in (Conen and Sandholm, 2001) based on the rank lattice, which exploits the topological structure of combinatorial auctions and the rank information of bidders' valuation. Typical query types used in these frameworks are value queries which directly ask a bidder its value on a bundle, order queries which ask a bidder to rank two bundles, and bound-approximation queries which ask a bidder to compute an upper/lower bound on a bundle.

Another type of preference elicitation approach is the ascending combinatorial auctions (Gul and Stacchetti, 2000; Demange et al., 1986; Kelso Jr and Crawford, 1982; Ausubel and Milgrom, 2002; Parkes and Ungar, 2000). These approaches elicit bidders' preferences by associating prices on each item or bundle. The prices are continuously updated during the elicitation process, and can only increase. The bidders are required to answer the demand queries (Blumrosen and Nisan, 2009), by submitting bids that maximize its utility under certain prices.

Preference elicitation approaches can effectively reduce the evaluation burden of bidders. However, normally researchers do not consider the hardness for a bidder in answering the queries. When applied to real problems, such as DRCMPSP, a bidder may find that it still cannot answer some queries even after incurring significant computation costs. Several works in costly preference elicitation (Baarslag and Gerding, 2015; Parkes, 2005; Larson and Sandholm, 2001) explicitly consider the cost model of a bidder in answering queries, but real bidders usually do not own these cost models. In contrast, our approach accepts approximate answers to the capacity queries, which can be generated very efficiently using approximation algorithms (e.g. the one we developed in Appendix A). To summarize, compared with the typical preference elicitation approaches, our approach focuses on how to rapidly find a good allocation of the DRCMPSP auction, a real-world application of combinatorial auction. On the other hand, our approach does not exclude the possibilities of integrating with existing preference elicitation methods, including the frameworks and query types, to further improve the results.

2.3 Techniques for Proactive Scheduling

Considerable amounts of works have been done for planning and scheduling under uncertainty. Several surveys (Bruni et al., 2015; Bidot et al., 2009; Herroelen and

Leus, 2005) are available for a complete review of the methodologies in this field. In this section, we focus on reviewing existing works on proactive scheduling, which are closely related to our research. Specifically, we classify existing approaches according to their solution types, following the taxonomy in (Bidot et al., 2009).

The first category of proactive scheduling approaches adopts start-time schedules as solutions. The optimization objective is often risk-aware, which is to find a schedule with the minimal makespan and has a high chance of being feasible during actual execution. State-of-the-art approaches in this category (Lamas and Demeulemeester, 2016; Varakantham et al., 2016) achieve the risk-aware optimization by adding a probabilistic constraint to the Mixed Integer Linear Program (MILP) for the deterministic RCPSP. The additional constraint can guarantee that the probability of schedule violation is restricted to a certain risk level, but the resulting probabilistic-constrained MILP models are hard to solve and need to be tackled by sampling based methods. These approaches are not applicable when the duration uncertainty is time-dependent, since the duration samples cannot be obtained without knowing the activity start times.

Another type of solutions for proactive scheduling is flexible solutions. Different from start-time schedules, flexible solutions make part of decisions before execution, and complete solutions (i.e. start time schedules) can be obtained according to the actual execution situations. The dynamically controllable Simple Temporal Network with Uncertainty (STNU) (Cui et al., 2015; Morris et al., 2001; Morris and Muscettola, 2005) is a typical example of flexible solution. However, STNU-based approaches are not directly applicable to resource-constrained scheduling problems, since they often focus on temporal reasoning only. Redundancy-based techniques (Davenport et al., 2001; Lambrechts et al., 2011) for machine breakdowns are another type of approach that generate flexible solutions by protecting activities using extra temporal slacks. The uncertainty model of machine breakdown is similar to our problem, since the breakdown probability is often time-related. However, these approaches are heuristic solutions, and are limited to specific probability distributions of machine breakdowns (e.g. normal (Davenport et al., 2001) and exponential (Lambrechts et al., 2011)). In contrast, our approach is built on principled approximation scheme, and does not require the stochastic knowledge to follow certain distributions.

For RCPSP, perhaps the most commonly used type of flexible solution is Partial-Order Schedule (POS). In (Policella et al., 2004), two approaches that directly generate POS from a deterministic RCPSP instance are proposed, including (a) Envelop Based Algorithm (EBA) and (b) Earliest Start Time Algorithm (ESTA). We will present more details about these two approaches in Section 4.6.1. A desirable property of these approaches is that they can be applied to handle any type of duration uncertainty, since they do not require any stochastic knowledge. However, when the stochastic knowledge is available, it is of great advantage to exploit it to generate significantly better solutions. More recently, several approaches are proposed to generate POS based on known stochastic knowledge (Beck and Wilson, 2007; Fu et al., 2012, 2015, 2016), for optimizing the robust makespan. However, a major assumption in these works is that the probability models of activity durations are independent of their start times. Therefore, they cannot be applied to solve our proactive scheduling problem.

The last type of flexible solution mentioned in (Bidot et al., 2009) is conditional schedule, referring to solutions with alternative branches of decisions. Selection of branches will be made contingently based on the actual execution. For the traditional stochastic RCPSP, which is a risk-neutral proactive scheduling problem with time-independent duration uncertainties, a number of approaches have been proposed to optimize the so-called elementary policies, which are essentially conditional schedules (Igelmund and Radermacher, 1983; Möhring, 2000; Stork, 2001; Ballestín, 2007; Ashtiani et al., 2011; Creemers, 2015). The current best approach for stochastic RCPSP is the dynamic programming procedure in (Creemers, 2015). Our proactive approaches different from this work in several aspects. Firstly, elementary policy generalizes POS. Essentially, an elementary policy starts the activities at the completion time of some other activity (equivalent to adding a precedence constraint), based on the actual execution of the project. But the resulting temporal network of an elementary policy is not necessarily a POS, since the resource conflicts can only be resolved for the actual execution scenario, instead of for all the possible scenarios. Therefore, POS can be considered as a special case of elementary policy, hence an optimal elementary policy could have better expected makespan than an optimal POS. However, the solution space of elementary policy is much larger than that of POS. In addition, the memory requirements for computing and

storing the conditional schedules are very high (Bidot et al., 2009). Secondly, the dynamic programming procedure in (Creemers, 2015) can only terminate when the optimal policy is found, while our algorithms are anytime and can be terminated early with high-quality feasible solutions. Finally, our approach is applicable to the time-dependent workability uncertainty, while the approach in (Creemers, 2015) can only solve the stochastic RCPSP with time-independent uncertainties.

2.4 Summary

In this chapter, we first give a brief introduction on the typical approaches for traditional centralized and deterministic project scheduling problems. Further, we review existing methodologies for the research problems we studied in this thesis. For DRCMPSP, though combinatorial auction is an ideal paradigm for designing distributed approaches, existing works only apply to simple problems with single-unit resources and small numbers of projects and activities. In contrast, the auction based approach we designed in Chapter 3 scales well to much larger problems with multi-unit resources, with superior solution quality compared with state-of-the-art approaches. For proactive scheduling, current methodologies can only handle problems with random activity durations that have no relation with activity start times. In Chapters 4 and 5, we study the risk-neutral and risk-aware proactive scheduling problems with time-dependent workability uncertainty, which cannot be solved by previous approaches. We formulate the corresponding stochastic optimization problems that generalize the traditional time-independent versions, and design several efficient algorithms by exploiting interesting problem structures.

Chapter 3

Distributed Multi-Project Scheduling based on Multi-Unit Combinatorial Auction

In this chapter, we focus on how to solve DRCMPSP using combinatorial auction. Though this is not a new paradigm, previous combinatorial auction based approaches only apply to simple problems with single-unit resources and small problem instances. In this chapter, we design an effective approach that works well on complex scheduling problems with multi-unit resources and large numbers of projects and activities. We first formulate the scheduling problem as a multi-unit combinatorial auction, where the shared global resources are auctioned to the autonomous project agents. We then formally analyze the connections between the scheduling problem and auction problem and prove the correctness of our formulation, i.e. an optimal schedule can be found by finding an optimal resource allocation under reasonable assumption. Nevertheless, the combinatorial auction formulation brings challenge to the project agents since the valuation problem is computationally intractable. We resolve this issue by introducing capacity query, which can efficiently elicit useful information from the project agents. Based on capacity query, we propose two strategies that can efficiently find high-quality allocations, which also lead to high-quality schedules. Our empirical results show that the two strategies scale well to large problem instances with tens of projects and thousands of activities, and can give better schedules than state-of-the-art DRCMPSP approaches.

The following content in this chapter is organized as follows. In Section 3.1, we give the formal statement of DRCMPSP. In Section 3.2, we present the combinatorial auction formulation for DRCMPSP, and theoretically analyze the relation between the scheduling and auction problems. In Section 3.3, we describe our methods for efficiently solving the DRCMPSP auction. Empirical evaluation is provided in Section 3.4, and the chapter is concluded in Section 3.5.

3.1 Problem Statement

We first give a formal statement of RCPSP. An instance of deterministic RCPSP involves a set of activities $A = \{a_1, ..., a_N\}$ that needs to be scheduled in a horizon of T consecutive time slots, and a set of renewable resources $R = \{r_1, ..., r_K\}$ where each $r_k \in R$ has a finite capacity of $C_k \in \mathbb{N}$ units in each time slot. Each activity $a_i \in A$ has a fixed duration of $d_i \in \mathbb{N}$ time slots, and requires $b_i^k \in \mathbb{N}$ units of resource r_k in each time slot of its processing duration. Throughout this thesis, we follow the common assumption in RCPSP that the activities are non-preemptive, meaning that once started they cannot be stopped until completion. For deterministic RCPSP, this means if activity a_i starts at time s_i, then its completion time is $c_i = s_i + d_i$. A project may have an earliest start time ed, which constraints the start times of all activities to be no earlier than ed. For single project problems, ed is often set to 0. For convenience, usually two dummy activities a_0 and a_{N+1} with zero durations and resource requirements are added to represent the start and completion of the project. A pair of activities a_i and a_j in $A_p = A \cup \{a_0, a_{N+1}\}$ could have a precedence relation $a_i \prec a_j$, which is a temporal constraint indicating that a_j must start after the completion of a_i. Let $E_p = \{(a_i, a_j) | a_i \prec a_j, \forall a_i, a_j \in A_p\}$ be the set of all precedence constraints, and $Pre(a_i) = \{a_j \in A_p | a_j \prec a_i\}$ be the immediate predecessors of an activity a_i.

A solution to a deterministic RCPSP instance is a (start-time) schedule, which is a vector $S = (s_0, ..., s_{N+1})$, where s_i is the start time of a_i. The completion time of the project specified by S is $ct(S) = \max\{c_i | a_i \in A_p\}$, then the *makespan* of S is $MS(S) = ct(S) - ed$. A feasible schedule must satisfy all the resource and temporal constraints. Let $\boldsymbol{S}$ be the set of all feasible schedules. Then to solve the deterministic RCPSP, we need to find an optimal solution $S^* \in \boldsymbol{S}$ such that certain

criterion is optimized. The most commonly used criterion for evaluating a schedule S is the minimization of makespan. When all the resource constraints are relaxed, i.e. resource capacities are unlimited, the makespan minimization problem can be solved very efficiently in polynomial time using the well-known Critical Path Method (Kelley Jr, 1961). This makespan, denoted as CPL, is also the minimum possible makespan regardless of resource capacities. However, when the resource capacities are limited, this problem is proved to be strongly NP-hard (Blazewicz et al., 1983).

In some applications, the project has an (expected) due date dd that represents a soft constraint can be violated with some cost. In this case, the optimization objective could be the minimization of *project delay*, defined as $dl(S) = \max\{0, ct(S) - dd\}$. Specifically, when the due date is set to $ed + CPL$, minimizing project delay is equivalent to minimizing makespan since $MS(S) - CPL \geq 0$ for any $S \in \boldsymbol{S}$.

Next, we state DRCMPSP. An instance of this problem involves M projects to be scheduled in a horizon of T consecutive time slots. Each project P_i, $i \in \{1, ..., M\}$, has its own activity set $A_i = \{a_{ij} | 1 \leq j \leq N_i\}$.[1] Two activities in the same project P_i could have a precedence relation. Each project P_i has its own earliest start time ed_i and an expected due date dd_i. For each P_i, a Project Agent PA_i is assigned to control P_i. Each PA_i can receive a revenue rv_i upon the completion of P_i. Let $\boldsymbol{P} = \{P_1, ..., P_M\}$ be the set of all projects.

A DRCMPSP instance also involves K types of renewable global resources. A global resource is denoted as r_k, $k \in \{1, ..., K\}$, with a limited capacity C_{kt} in each time slot $t \in \{1, .., T\}$. Then, the requirement of a_{ij} for r_k is denoted as b_{ij}^k. All global resources are managed by a Mediator Agent MA. Meanwhile, a_{ij} may also require certain amounts of local resources that are owned and fully controlled by PA_i. A local resource is denoted as r_{l_i}, $l_i \in \{1, ..., L_i\}$, with a limited capacity $C_{l_i t}$ in time slot t. Then, the requirement of a_{ij} for r_{l_i} is denoted as $b_{ij}^{l_i}$.

A solution of a DRCMPSP case is a multi-project schedule $\boldsymbol{\mathcal{S}} = (S_1, ..., S_M)$ with S_i being the schedule of project P_i, and $S_i = (s_{i1}, ..., s_{iN_i})$ with s_{ij} being the start time of activity a_{ij}. A solution $\boldsymbol{\mathcal{S}}$ is *feasible*, if all the hard constraints are satisfied (i.e. the earliest start times, precedence relations and resource constraints are respected). Denote the set of feasible solutions as $\boldsymbol{\mathcal{S}}$. The soft constraints (i.e.

[1]For DRCMPSP, we will use i and j to index projects and activities, respectively.

expected due dates), on the other hand, are used in evaluating the quality of a feasible solution $\mathcal{S} \in \boldsymbol{\mathcal{S}}$. For project P_i, its delay specified by the schedule S_i in $\mathcal{S}$ is written as $dl_i(S_i) = \max\{0, ct(S_i) - dd_i\}$, similar to the definition of RCPSP.

In previous research, the most commonly used criterion for evaluating a solution $\mathcal{S}$ is the minimization of *Average Project Delay* (APD):

$$APD(\mathcal{S}) = \frac{1}{M} \sum_{i=1}^{M} dl_i(S_i). \tag{3.1}$$

However, APD cannot reflect the heterogeneity of projects. It is quite common in practice that different projects exhibit different degrees of importance. In this case, the minimization of *Weighted Project Delay* (WPD) is more reasonable:

$$WPD(\mathcal{S}) = \frac{1}{M} \sum_{i=1}^{M} w_i \cdot dl_i(S_i), \tag{3.2}$$

where w_i is the weight of project i. When the weights of all projects are the same, WPD is equivalent to APD. Usually w_i is considered as the (monetary) unit penalty for any delay of project i. Hence, when the delay cost functions $dc_i(S_i)$ for all projects are linear, i.e. $dc_i(S_i) = w_i \cdot dl_i(S_i)$, the minimization of WPD is equivalent to the minimization of *Total Delay Cost* (TDC):

$$TDC(\mathcal{S}) = \sum_{i=1}^{M} dc_i(S_i) = M \cdot WPD(\mathcal{S}). \tag{3.3}$$

It should be noted that when the delay cost is not linear, TDC is not equivalent to WPD. However, we argue that TDC is more practical than WPD and APD, since it is a general monetary measurement on the solution quality. In Section 3.2 we will show that our combinatorial auction formulation corresponds to the minimization of TDC, under the condition that revenues of the projects are sufficiently high. Since TDC does not rely on the form of $dc_i(S_i)$, our approach can also adopt nonlinear delay cost functions, as long as they satisfy the property of monotonicity which fits the common sense of delay, i.e. for two schedules S_i^1 and S_i^2 of P_i, if $ct(S_i^1) > ct(S_i^2)$ holds for P_i, then $dc_i(S_i^1) > dc_i(S_i^2)$.

Below we give a formal definition of the optimization problem that needs to be solved for a DRCMPSP case:

Definition 3.1 *(The optimization problem of DRCMPSP) Given a DRCMPSP case where the project set* $\boldsymbol{P}$ *and global resource set* R *are specified, find a solution* $\mathcal{S}^* \in \boldsymbol{\mathcal{S}}$ *such that a chosen criterion (e.g. APD, WPD, TDC) is optimized.*

Regarding information privacy, variables and parameters of activities (e.g. start times s_{ij}, durations d_{ij}, resource requirements b_{ij}^{k} and $b_{ij}^{l_i}$, and precedence relations) and local resources (e.g. each local resource r_{l_i} and capacity profiles $C_{l_i t}$) are considered as private to each project agent.

3.2 Multi-Unit Combinatorial Auction Formulation of DRCMPSP

In this section, we describe how to transform DRCMPSP to a multi-unit combinatorial auction problem. We first give a description of multi-unit combinatorial auction. Suppose that in the auction, there are m items to be sold to n bidders. Each item I_e, $e \in \{1, ..., m\}$ has a limited capacity of C_e units (copies). We call the vector $C = (C_1, ..., C_m)$ as the *item capacity profile*. Each bidder can express its preference on any *multiset* (Krysta et al., 2013) of items, defined as a vector $\Lambda = (\lambda_1, ..., \lambda_m)$, where λ_e is the required units of item I_e. Here we use $\lambda(\Lambda, e)$ to denote the eth element of a specific multiset Λ. Then, given an item capacity profile C, the set of all the possible multisets is denoted as $\mathbf{\Lambda}(C) = \left\{\Lambda \middle| 0 \leq \lambda(\Lambda, e) \leq C_e, \forall e \in \{1, ..., m\}\right\}$. An *allocation* of the items is a vector $O = (O_1, ..., O_n)$, where $O_f \in \mathbf{\Lambda}(C)$ is a multiset allocated to bidder f, $f \in \{1, ..., n\}$. An allocation is said to be feasible if it respects the item capacity profile, i.e. $\sum_{f=1}^{n} \lambda(O_f, e) \leq C_e$, $\forall e \in \{1, ..., m\}$. Denote the set of all feasible allocations as $\boldsymbol{O}(C)$.

For each $\Lambda \in \mathbf{\Lambda}(C)$, a bidder f has a *value* denoted as $v_f(\Lambda)$, which indicates the maximum monetary amount that f is willing to pay for Λ. Thus we can define a valuation function $v_f : \mathbf{\Lambda}(C) \mapsto \mathbb{R}^+$ for a bidder f. Knowing all the valuation functions $\boldsymbol{v} = (v_1, ..., v_n)$, the auctioneer computes an allocation $O^* \in \boldsymbol{O}(C)$ to maximize the *social welfare*, defined as $SW(O) = \sum_{f=1}^{n} v_f(O_f)$ for a given allocation O. It also computes a vector $p(O^*) = (p_1(O_1^*), ..., p_n(O_n^*))$, where $p_f(O_f^*)$ is the *payment* for f for obtaining O_f^*. Based on the computation results of the auctioneer, a bidder f can gain a *utility* $u_f(O_f^*) = v_f(O_f^*) - p_f(O_f^*)$. Below we give a formal definition of multi-unit combinatorial auction based on the above notations:

Definition 3.2 *(Multi-unit combinatorial auction) Given the item capacity profile C and valuation function set $\boldsymbol{v}$, compute an allocation $O^* \in \boldsymbol{O}(C)$ and payments*

$p(O^*)$, such that $SW(O^*)$ is the maximal social welfare. In other words,

$$O^* = \underset{O \in \boldsymbol{O}(C)}{argmax}\ SW(O). \tag{3.4}$$

Note that the problem of finding the optimal allocation O^* is NP-complete (Sandholm, 2002).

Next, we formulate DRCMPSP as a multi-unit combinatorial auction. We focus on the allocation problem in this section, i.e. the problem of finding O^*, while the payment schemes will be discussed in Section 3.3. Following the previous combinatorial auction based approaches, here PAs act as bidders, the MA acts as the auctioneer, and the goods to sell are the global resources at each time slot. Hence, given a DRCMPSP case consists of M PAs, K types of global resources with horizon T, there will be M bidders to compete for $K \times T$ items in the corresponding auction. For convenience, here we denote an item and its capacity as I_{kt} and C_{kt}, respectively. Then, the item capacity profile and multiset are matrices instead of vectors, denoted as $C = [C_{kt}]_{K \times T}$ and $\Lambda = [\lambda_{kt}]_{K \times T}$, respectively, and the requirement of resource r_k on time t specified in a multiset Λ is denoted as $\lambda(\Lambda, k, t)$. Next, we discuss how a bidder evaluates a multiset Λ, i.e. computes $v_i(\Lambda)$. We begin with the following definition:

Definition 3.3 *(Feasible multiset) A schedule S_i is feasible to project P_i if all hard constraints of P_i (including earliest start time, precedence constraints, and local/global resource constraints) are satisfied. A multiset Λ is* feasible *to PA_i if there exists a feasible schedule S_i of P_i when PA_i obtains the amount of global resources specified in Λ.*

Now we can define the valuation function of PA_i as follows:

$$v_i(\Lambda) = \begin{cases} \max\{rv_i - DC_i(\Lambda), 0\}, & \text{if } \Lambda \text{ is feasible} \\ 0, & \text{otherwise,} \end{cases} \tag{3.5}$$

where $DC_i(\Lambda)$ is the delay cost by obtaining Λ. Note that when the revenue rv_i is lower than the delay cost of a feasible multiset Λ, PA_i will not use it to complete its project since it will cause him a negative utility, though a feasible schedule exists. Therefore, in this case, PA_i gives a value of 0 to Λ. The following definition is used to show how to determine $DC_i(\Lambda)$:

Definition 3.4 *(Solution set and primal schedule) Given a feasible multiset* Λ *of* PA_i*, the set* $\boldsymbol{S}_i(\Lambda)$ *which contains all feasible schedules of* P_i *given the amounts of global resources specified in* Λ*, is called a* solution set*. A solution* $S_i^*(\Lambda) \in \boldsymbol{S}_i(\Lambda)$ *that has the minimum completion time* $ct(S_i^*(\Lambda))$ *is called a* primal schedule *of* Λ *to* PA_i*.*

Note that there could be more than one primal schedule in a solution set, but these primal schedules result in the same delay cost, according to Definition 3.4. We define the delay cost $DC_i(\Lambda)$ as the delay cost of the primal schedule, i.e.

$$DC_i(\Lambda) = \min \{dc_i(S_i) | S_i \in \boldsymbol{S}_i(\Lambda)\} = dc_i(S_i^*(\Lambda)). \tag{3.6}$$

According to Equation (3.5) and (3.6), PA_i can evaluate a given multiset Λ using the following steps: 1) check if it is feasible and 2) if feasible, find a primal schedule. Then, the complete valuation function $v_i(\Lambda)$ can be built by computing the value of each possible multiset $\Lambda \in \boldsymbol{\Lambda}(C)$. After collecting the complete valuation functions from all PAs, the MA can find the optimal allocation O^* which maximizes the social welfare, as stated in Definition 3.2. We call the auctions formulated this way as DRCMPSP auctions. It should be noted that the valuation functions of a PA could exhibit both *complementarity* and *substitutability* (Sandholm, 2002). A simple example is shown in Figure 3.1, where a project P_1 containing only one activity a_{11} needs to be scheduled onto a single global resource with capacity 3 in each time slot of the horizon $T = 5$. P_1 has a revenue of 10, and a_{11} has a duration of 2 and requires 1 unit of the global resource. The right part of Figure 3.1(a) shows a feasible schedule S_1 where a_{11} is scheduled to start at $t = 1$. In Figure 3.1(b), it is easy to see that the two multisets Λ_1 and Λ_2 are infeasible thus $v_1(\Lambda_1) = v_1(\Lambda_2) = 0$, but the multiset $\Lambda_3 = \Lambda_1 + \Lambda_2$ is feasible and has a value of $v_1(\Lambda_3) = 8$. Therefore, Λ_1 and Λ_2 in Figure 3.1(b) have complementarity since $v(\Lambda_1) + v(\Lambda_2) \leq v(\Lambda_1 + \Lambda_2)$. On the contrary, the multisets Λ_1, Λ_1 and $\Lambda_3 = \Lambda_1 + \Lambda_2$ in Figure 3.1(c) are feasible and $v_1(\Lambda_1) = v_1(\Lambda_2) = v_1(\Lambda_3) = 8$, therefore Λ_1 and Λ_2 in Figure 3.1(c) have substitutability since $v(\Lambda_1) + v(\Lambda_2) \geq v(\Lambda_1 + \Lambda_2)$.

Next, we analyze the relation between the optimal allocation O^* of the auction and the optimal solution $\mathcal{S}^*$ of the corresponding DRCMPSP case with the objective of minimizing TDC as defined in Equation (3.3). We first introduce the *revenue condition* in the following definition:

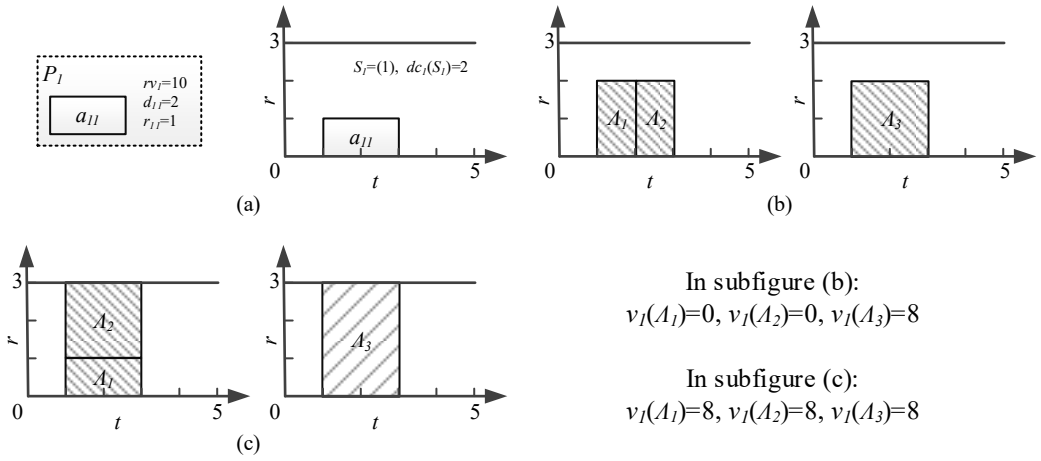


Figure 3.1: An Example of Complementarity and Substitutability

Definition 3.5 *(Revenue condition) Given a feasible solution $\mathcal{S} \in \boldsymbol{\mathcal{S}}$ of a DRCMPSP case, let $dc^*(\mathcal{S}) = max\{dc_i(S_i)|i \in \{1, ..., M\}\}$ be the highest delay cost of a PA in $\mathcal{S}$. Let $\boldsymbol{\mathcal{S}'} \subseteq \boldsymbol{\mathcal{S}}$ be a set of feasible solutions. Denote $dc^\dagger(\boldsymbol{\mathcal{S}'}) = min\ \{dc^*(\mathcal{S})|\mathcal{S} \in \boldsymbol{\mathcal{S}'}\}$. If $\forall i \in \{1, ..., M\}$, $rv_i > dc^\dagger(\boldsymbol{\mathcal{S}'}) \cdot M$, then the DRCMPSP case satisfies the revenue condition of $\boldsymbol{\mathcal{S}'}$.*

Intuitively, the revenue condition states that for a subset $\boldsymbol{\mathcal{S}'}$ of solutions, the revenue rv_i of each PA i is larger than M times the minimum $dc^*(\mathcal{S})$ value, which is the largest project delay cost specified by a solution $\mathcal{S} \in \boldsymbol{\mathcal{S}'}$. Based on the revenue condition, the following theorem can be proved:

Theorem 3.1 *If a DRCMPSP case satisfies the revenue condition of $\boldsymbol{\mathcal{S}}$, then any optimal allocation O^* that maximizes the social welfare of the corresponding DRCMPSP auction produces an optimal solution $\mathcal{S}^*$ for the DRCMPSP case with the objective of minimizing TDC.*

Proof. Since the revenue condition is satisfied, there must exist a solution $\mathcal{S} \in \boldsymbol{\mathcal{S}}$ with $dc^*(\mathcal{S}) = dc^\dagger(\boldsymbol{\mathcal{S}})$. Let $O(\mathcal{S})$ be an allocation that can incorporate $\mathcal{S}$, i.e. $\forall i$, $S_i = S_i^*(O_i(\mathcal{S}))$. When the revenue condition is satisfied, each PA has a positive value for the multiset allocated to it in $O(\mathcal{S})$, since for all PA_i, we have $rv_i - DC_i(O_i(\mathcal{S})) = rv_i - dc_i(S_i) > dc^*(\mathcal{S}) \cdot M - dc^*(\mathcal{S}) \geq dc^*(\mathcal{S}) \cdot (M-1) \geq 0$. Therefore, $SW(O(\mathcal{S})) = \sum_{i=1}^{M}(rv_i - dc_i(S_i))$. We prove the theorem by the following steps.

Firstly, we show that allocating any PA a multiset with zero value can only produce a lower social welfare than $SW(O(\mathcal{S}))$. Assume there exists an allocation

O' which produces an equal or higher social welfare than $O(\mathcal{S})$ and allocates PA_j a multiset with zero value, i.e. $SW(O') \geq SW(O(\mathcal{S}))$ and $v_j(O'_j) = 0$. The maximum possible social welfare of $SW(O')$ is $SW^m = \sum_{i=1,i\neq j}^{M} rv_i$. If $SW^m \geq SW(O(\mathcal{S}))$, then $\sum_{i=1,i\neq j}^{M} rv_i \geq \sum_{i=1}^{M}(rv_i - dc_i(S_i))$, which leads to $rv_j \leq \sum_{i=1}^{M} dc_i(S_i)$. However, from the revenue condition, we know that $rv_j > dc^*(\mathcal{S}) \cdot M \geq \sum_{i=1}^{M} dc_i(S_i)$, which is a contradiction. Therefore, it is not possible to achieve an equal or higher social welfare than $SW(O(\mathcal{S}))$ by allocating a multiset with zero value to any PA.

Secondly, we show that an optimal solution $\mathcal{S}^*$ to the DRCMPSP case lead to an optimal allocation of the corresponding auction. Let $O(\mathcal{S}^*)$ be an allocation that can incorporate $\mathcal{S}^*$, i.e. $\forall i,\ S_i^* = S_i^*(O_i(\mathcal{S}^*))$. From the first step we know that in $O(\mathcal{S}^*)$, each bidder should be allocated a multiset with positive value, since otherwise it produces a lower value than $SW(O(\mathcal{S}))$ hence cannot be optimal. Therefore, $SW(O(\mathcal{S}^*)) = \sum_{i=1}^{M}(rv_i - dc_i(S_i^*))$. Assume there is another allocation O'' which grants all PAs feasible multisets with positive values, and produces a higher social welfare than $SW(O(\mathcal{S}^*))$. Denote the solution results from O'' as $\mathcal{S}'' = (S_1'', ..., S_N'')$, where $S_i'' = S_i^*(O_i'')$. Since $SW(O'') > SW(O(\mathcal{S}^*))$, $\sum_{i=1}^{M}(rv_i - dc_i(S_i'')) > \sum_{i=1}^{M}(rv_i - dc_i(S_i^*))$, indicating that $TDC(\mathcal{S}'') < TDC(\mathcal{S}^*)$ which contradicts the fact that $\mathcal{S}^*$ minimizes TDC. Hence, $O(\mathcal{S}^*)$ maximizes the social welfare of the DRCMPSP auction.

Finally, we show that any optimal allocation O^* of the DRCMPSP auction produces a solution of DRCMPSP with the same TDC value as $\mathcal{S}^*$. According to the first step, O^* should allocate each PA a feasible multiset O_i^* with positive value. Denote the solution results from O^* as $\mathcal{S}^O = \{S_1^O, ..., S_M^O\}$, where $S_i^O = S_i^*(O_i^*)$, therefore $SW(O^*) = \sum_{i=1}^{M}(rv_i - dc_i(S_i^O))$. According to the second step, $SW(O^*) = SW(O(\mathcal{S}^*))$, therefore $TDC(\mathcal{S}^O) = TDC(\mathcal{S}^*)$. □

Theorem 3.1 shows that when the revenue of each PA is sufficiently high (such that the revenue condition of $\boldsymbol{\mathcal{S}}$ is satisfied), solving DRCMPSP auction optimally will lead to an optimal solution of the DRCMPSP case. Note that when the revenue condition is not satisfied, the DRCMPSP auction can still work, but some projects may not be scheduled in the optimal allocation; in this case, the APD, WPD and TDC objectives are only effective for the projects that are scheduled. This makes sense because it is possible that when the resource capacities are very limited, a better choice is to serve a subset of the demands with a higher total (monetary)

value, instead of serving all demands but resulting in a lower total value.

In practice, suboptimal algorithms, instead of complete algorithms, are often used since the problem of finding O^* in Equation (3.4) is NP-complete. Regarding the relation between suboptimal allocations and solutions, we can have another conclusion following a similar proof as Theorem 3.1:

Corollary 3.1 *For a DRCMPSP case with the objective of minimizing TDC, let O^δ be a suboptimal allocation of the DRCMPSP auction with optimal allocation O^*, where $SW(O^*) - SW(O^\delta) = \delta$. Let $\boldsymbol{\mathcal{S}}^\delta = \{\mathcal{S} \in \boldsymbol{\mathcal{S}} | TDC(\mathcal{S}) - TDC(\mathcal{S}^*) \geq \delta\}$, where $\mathcal{S}^*$ is the optimal solution of the DRCMPSP case. If the revenue condition of $\boldsymbol{\mathcal{S}}^\delta$ is satisfied, then any allocation O with $SW(O) = SW(O^\delta)$ produces a suboptimal solution $\mathcal{S}^\delta$ of the DRCMPSP case, where $TDC(\mathcal{S}^\delta) - TDC(\mathcal{S}^*) = \delta$.*

Proof. As in the proof of Theorem 3.1, there must exist a solution $\mathcal{S} \in \boldsymbol{\mathcal{S}}^\delta$ where $dc^*(\mathcal{S}) = dc^\dagger(\boldsymbol{\mathcal{S}}^\delta)$. Therefore, according to the same reason in the first step in proving Theorem 3.1, allocating any PA a multiset with zero value can only produce a lower social welfare than $SW(O(\mathcal{S}))$, where $\forall i, S_i = S_i^*(O_i(\mathcal{S}))$. Next, for a solution $\mathcal{S}^\delta \in \boldsymbol{\mathcal{S}}^\delta$ with $TDC(\mathcal{S}^\delta) - TDC(\mathcal{S}^*) = \delta$, we construct an allocation $O(\mathcal{S}^\delta)$ that can incorporate $\mathcal{S}^\delta$, i.e. $\forall i, S_i^\delta = S_i^*(O_i(\mathcal{S}^\delta))$. Follow the second step in proving Theorem 3.1, we can see that $SW(O(\mathcal{S}^\delta)) = SW(\mathcal{S}^\delta)$ is the maximum social welfare that can be achieved by solutions in $\boldsymbol{\mathcal{S}}^\delta$. This is because if there is any allocation O'' with $SW(O'') > SW(O^\delta)$, we have $TDC(\mathcal{S}'') < TDC(\mathcal{S}^\delta)$, where $S_i'' = S_i^*(O_i'')$, hence $\mathcal{S}'' \notin \boldsymbol{\mathcal{S}}^\delta$. Finally, we can show that any allocation O that has the same social welfare as $O(\mathcal{S}^\delta)$ produces a solution with the same TDC value as $\mathcal{S}^\delta$, following similar procedure in the third step in proving Theorem 3.1. □

Note that the revenue conditions in Theorem 3.1 and Corollary 3.1 are both sufficient conditions. Also noted that the revenue condition of Corollary 3.1 is stronger than that of Theorem 3.1, since $\boldsymbol{\mathcal{S}}^\delta \subseteq \boldsymbol{\mathcal{S}}$. Therefore, when suboptimal algorithms are used in the DRCMPSP auction, the revenue requirement for each bidder should be higher than when complete algorithms are used, such that the suboptimality can be tolerated to produce solutions where all bidders are scheduled. In the following part of this chapter, we assume that the PAs' revenues are high enough for our (approximate) approaches to find a suboptimal solution where all projects are scheduled.

Till now, we have shown how to formulate DRCMPSP as a multi-unit combinatorial auction. In a DRCMPSP auction, all information required from the PAs is the valuation function v_i. Therefore, the PAs do not have to reveal any information about the activities and local resources. Moreover, it could be very hard to compute the private variables and parameters, even with the full knowledge of the valuation function. However, there exist several issues regarding the computational complexity. Firstly, as we mentioned, the allocation problem of combinatorial auction is NP-complete. In addition, in the worst case, to compute the optimal allocation, the bidders are required to convey their full valuations on every possible multiset to the auctioneer, which results in exponential growth of communication requirements. This is even worse in the DRCMPSP auction, since it could be intractable for a bidder to evaluate a given multiset. The main reason for the hardness on valuation is that the bidders need to deal with RCPSP, which is NP-hard. To be specific, firstly in Equation (3.5) it is hard to determine if a schedule is feasible (equivalent to solving a feasibility problem of RCPSP which is NP-hard (Neumann et al., 2012)), and secondly in Equation (3.6) it is hard to find the primal solution (equivalent to solving a RCPSP which is NP-hard (Blazewicz et al., 1983)). Due to the above reasons, it is impractical to require bidders to have full knowledge of their valuation functions. Instead, a viable way is to guide the bidders to focus on evaluating some multisets that are more useful for determining a good allocation. In this work, we introduce the capacity query to efficiently elicit valuations from bidders.

3.3 Solving the DRCMPSP Auction using Capacity Queries

This section details our approach. We first introduce the capacity query in Section 3.3.1. Then, in Section 3.3.2 and 3.3.3, we describe two strategies, the greedy allocation and the branch-and-bound heuristic, which are designed based on the ones introduced in (Gonen and Lehmann, 2000) and can be used with capacity query to solve the DRCMPSP auctions.

3.3.1 Capacity Query

Capacity queries try to use different item capacity profiles to elicit bidders' valuation on some multisets. The semantics of a capacity query is: "suppose the capacity profile is $\Psi = [\psi_{kt}]_{K\times T}$, which multiset you value the most, and what is your value on that multiset?". Mathematically, to answer a capacity query, a bidder needs to solve a *bidding problem*:

$$\Lambda_i^* = \underset{\Lambda\in\mathbf{\Lambda}(\Psi)}{\operatorname{argmax}}\ v_i(\Lambda), \tag{3.7}$$

where $\mathbf{\Lambda}(\Psi) = \{\Lambda | 0 \le \lambda(\Lambda, k, t) \le \psi_{kt}\}$. An answer to a capacity query is a tuple $B = \langle \Lambda, v_i(\Lambda)\rangle$, which is called a *bid*.

Before we show how a bidder can answer the capacity query in a DRCMPSP auction, we first define a special type of multisets that can be (approximately) evaluated easily. As we have mentioned in Section 3.2, it could be computationally intractable to determine the feasibility of a given multiset. On the other hand, generating a feasible multiset from a feasible schedule is quite easy, as stated in the following definition:

Definition 3.6 *Given a schedule $S_i = (s_{i1}, ..., s_{iN_i})$ of P_i, the multiset $\widehat{\Lambda}_i = \widehat{\Lambda}_i(S_i)$ calculated as:*

$$\lambda(\widehat{\Lambda}_i, k, t) = \sum_{a_{ij}\in A_i(t)} b_{ij}^k, \tag{3.8}$$

where $A_i(t) = \left\{a_{ij}, j \in \{1, ..., N_i\} \middle| s_{ij} \le t, (s_{ij} + d_{ij}) > t\right\}$, is called a core[2] *of PA_i.*

The following lemma shows two important properties of cores:

Lemma 3.1 *The following two statements hold for PA_i in a DRCMPSP auction: 1) for all cores, the numbers of total required item units are the same; 2) any multisets with lower total required item units than a core is infeasible.*

Proof. It is straightforward to see that these two statements hold. Firstly, it is easy to see that the number of total required item units $RQ(\widehat{\Lambda}_i)$ of a core $\widehat{\Lambda}_i$ is $RQ(\widehat{\Lambda}_i) = \sum_{k=1}^{K}\sum_{j=1}^{N_i} b_{ij}^k d_{ij}$. Since b_{ij}^k and d_{ij} are all fixed values, the first statement holds. We denote the number of total required item units of PA_i's cores as RQ_i. Secondly, it is also easy to see that for any multiset $\widetilde{\Lambda}$ with lower total required item units than $\widehat{\Lambda}_i$, there must be at least one activity which cannot receive enough global resource. Therefore, the second statement holds. □

[2]Note that this definition is different from the commonly used definition of core in game theory.

Lemma 3.1 indicates that a core of PA_i is a compact resource utilization profile of its project P_i. Next we show how to evaluate a core by proving the following lemma:

Lemma 3.2 *For PA_i, the value $v_i(\widehat{\Lambda}_i)$ of a core $\widehat{\Lambda}_i = \widehat{\Lambda}_i(S_i)$ generated from a feasible schedule $S_i = (s_{i1}, ..., s_{iN_i})$ can be approximated by $\widetilde{v}_i(\widehat{\Lambda}_i)$ with an error $err(S_i) = v_i(\widehat{\Lambda}_i) - \widetilde{v}_i(\widehat{\Lambda}_i)$, where $0 \leq err(S_i) \leq dc_i(S_i) - DC_i(\widehat{\Lambda}_i)$ and*

$$\widetilde{v}_i(\widehat{\Lambda}_i) = max\ \{rv_i - dc_i(S_i), 0\}. \tag{3.9}$$

Specifically, when no activity of project P_i requires only local resources, the approximation is exact, i.e. $v_i(\widehat{\Lambda}_i) = \widetilde{v}_i(\widehat{\Lambda}_i)$.

Proof. By definition, $v_i(\widehat{\Lambda}_i) \geq \widetilde{v}_i(\widehat{\Lambda}_i)$, since $S_i \in \boldsymbol{S}_i(\widehat{\Lambda}_i)$ which leads to $DC(\widehat{\Lambda}_i) \leq dc_i(S_i)$, therefore $err(S_i) \geq 0$. Further, when $DC(\widehat{\Lambda}_i) \leq dc_i(S_i) \leq rv_i$, $err(S_i) = dc_i(S_i) - DC(\widehat{\Lambda}_i)$; when $DC(\widehat{\Lambda}_i) \leq rv_i \leq dc_i(S_i)$, $err(S_i) = rv_i - DC(\widehat{\Lambda}_i) \leq dc_i(S_i) - DC(\widehat{\Lambda}_i)$; when $rv_i \leq DC(\widehat{\Lambda}_i) \leq dc_i(S_i)$, $err(S_i) = 0$. To sum up, $0 \leq err(S_i) \leq dc_i(S_i) - DC(\widehat{\Lambda}_i)$.

Next, we show that the approximation is exact when no activity of P_i requires only local resources. We only need to prove that S_i is the primal schedule of $\widehat{\Lambda}_i$. Assume there is another schedule $S_i' \in \boldsymbol{S}_i(\widehat{\Lambda}_i)$ that results in a smaller delay cost than S_i, i.e. $dc_i(S_i') < dc_i(S_i)$, therefore $ct(S_i') < ct(S_i)$. Denote the core generated from S_i' as $\widehat{\Lambda}_i' = \widehat{\Lambda}_i(S_i')$, then we have $\lambda(\widehat{\Lambda}_i', k, t) = 0$ for any $ct(S_i') < t \leq ct(S_i)$, which indicates that $\lambda(\widehat{\Lambda}_i', k, t) < \lambda(\widehat{\Lambda}_i, k, t)$. According to Lemma 3.1, to maintain feasibility, there must exist some $t \leq ct(S_i')$ such that $\lambda(\widehat{\Lambda}_i', k, t) > \lambda(\widehat{\Lambda}_i, k, t)$. Hence, S_i' cannot belong to the solution set of $\widehat{\Lambda}_i$, which is a contradiction. □

According to Lemma 3.2, the value of a core can be (approximately) evaluated very easily. Now we are ready to show how to solve the bidding problem.

Proposition 3.1 *For PA_i, given a capacity profile $\Psi = [\psi]_{K \times T}$, if the solution set $\boldsymbol{S}_i(\Psi)$ is not empty, [3] then the core $\widehat{\Lambda}_i(S_i^*)$ generated from the schedule S_i^* that has the minimum delay cost solves the bidding problem optimally.*

Proof. Consider Ψ as a multiset, then the solution set $\boldsymbol{S}_i(\Psi)$ includes all the possible schedules under Ψ, and S_i^* is the primal schedule of Ψ. For any feasible multiset

[3] When $S_i(\Psi) = \emptyset$, all multisets in $\boldsymbol{\Lambda}(\Psi)$ are infeasible. In that case, PA_i can simply bid for any $\Lambda \in \boldsymbol{\Lambda}(\Psi)$ with $B_i = \langle \Lambda, 0 \rangle$.

Λ' under Ψ, its delay cost must be equal to that of a schedule $S'_i \in \boldsymbol{S}_i(\Psi)$, hence $v_i(\Lambda') = \max\{rv_i - DC_i(\Lambda'), 0\} \leq \max\{rv_i - DC_i(\widehat{\Lambda}_i(S^*_i)), 0\} = v_i(\widehat{\Lambda}_i(S^*_i))$. For any infeasible multiset Λ'' under Ψ, $v_i(\widehat{\Lambda}_i(S^*_i)) \geq 0 = v_i(\Lambda'')$. Therefore, $v_i(\widehat{\Lambda}_i(S^*_i))$ is an optimal solution of the bidding problem. □

According to Proposition 3.1, on one hand, $\widehat{\Lambda}_i(S^*_i)$ is one of the optimal solutions to the bidding problem. Therefore, given a capacity query with profile Ψ, a bidder can find the answer by solving a RCPSP with time-varying global resource capacities as specified in Ψ. Note that for the optimal schedule S^*_i, the value of $\widehat{\Lambda}_i(S^*_i)$ given by Equation (3.9) is exact, since $dc(S^*_i)$ is the minimum delay cost for any $S \in \boldsymbol{S}_i(\Psi)$. We call the bid $B^*_i(\Psi) = \langle \widehat{\Lambda}_i(S^*_i), v_i(\widehat{\Lambda}_i(S^*_i))\rangle$ as the exact answer to the capacity query with capacity profile Ψ. On the other hand, however, $\widehat{\Lambda}_i(S^*_i)$ is not the only candidate for answering the query, since Equation (3.6) indicates that there could be some multisets, which are not cores, that result in the same delay cost hence the same value as $\widehat{\Lambda}_i(S^*_i)$ but request larger amounts of some items.[4] In Section 3.3.2 and 3.3.3, we will show that under certain payment schemes, it is sufficient for a bidder PA_i to answer the capacity queries with cores, which require the minimum amount of items RQ_i.

3.3.2 Greedy Allocation

In (Gonen and Lehmann, 2000), a greedy allocation strategy is developed for solving the basic winner determination problem (WDP) of multi-unit combinatorial auction. This problem aims at finding the winning bids from a set of collected bids, such that the sum of the winning bids' values is maximized. A heuristic *Average Price* is introduced in (Gonen and Lehmann, 2000), which estimates the "contribution" that a bid can make to the allocation. A bid is said to have a higher average price, if its value is higher and the number of its total required item units is lower. Any function that satisfies the above intuition can be used as average price. The most widely used one is the ratio of bid value to the square root of the total item units, i.e. for a bid $B = \langle \Lambda, v_i(\Lambda)\rangle$, the average price $\gamma(B)$ is calculated as

$$\gamma(B) = v_i(\Lambda) \Big/ \sqrt{\sum_{k=1}^{K}\sum_{t=1}^{T} \lambda(\Lambda, k, t)}\ . \tag{3.10}$$

[4] An example is the multiset Ψ, since S^*_i is the primal schedule of Ψ.

In (Gonen and Lehmann, 2000), Gonen and Lehmann show that when working with a greedy strategy, the average price function in Equation (3.10) produces a solution to the multi-unit WDP with a worst-case approximation ratio of $\sqrt{H}$, where H is the total available item units. This is the tightest theoretical bound on the approximation ratio that a (polynomial-time) greedy algorithm can achieve for a multi-unit WDP. The greedy allocation strategy in (Gonen and Lehmann, 2000) iteratively grants the bid that 1) does not conflict with the partial allocation and 2) has the maximum average price value in the remaining bids, until no bid can be granted.

In this section we describe how to use the greedy allocation strategy in (Gonen and Lehmann, 2000) to work with capacity queries to efficiently obtain a good allocation to the DRCMPSP auction. It should be noted that DRCMPSP auction is different from the basic WDP in (Gonen and Lehmann, 2000) even when the valuation functions are fully known by the PAs (which is not realistic due to the hardness of evaluation), because of the existence of substitutability (Sandholm, 2002). Essentially, the reason is that the basic WDP in (Gonen and Lehmann, 2000) does not distinguish bids from different bidders, which could result in an allocation where a bidder be granted multiple bids from it. Take the PA with the project shown in Figure 3.1(a) as an example, if it submits two bids for the two multisets Λ_1 and Λ_2 in Figure 3.1(c), then the auctioneer could allocate both Λ_1 and Λ_2 (i.e. Λ_3) to it with a value of 16, but in fact the PA only gets a value of 8. We make the greedy allocation strategy workable for the DRCMPSP auction, by granting only one bid from the same bidder. Later we will show that under a reasonable assumption, the guarantee of $\sqrt{H}$ for the basic WDP still holds for the DRCMPSP auction (in our case $H = \sum_{k=1}^{K} \sum_{t=1}^{T} C_{kt}$), if the capacity queries can be answered exactly, i.e. the bidding problems can be solved optimally.

The details of the greedy allocation strategy are shown in Algorithm 1. In general, the auctioneer maintains a set $\overline{US}$ of unallocated bidders, and iteratively asks each bidder in $\overline{US}$ a capacity query. After collecting the answers, i.e. bids, the auctioneer grants the bid with the maximum average price, and updates the capacity of each item. The allocated bidder will be removed from $\overline{US}$. This process continues until $\overline{US}$ is empty. Apparently, each bidder will only be allocated once for one of its bids. For payment calculation in Algorithm 1, here we adopt the fixed unit

Algorithm 1: Greedy allocation algorithm

Input: Bidder set $\boldsymbol{PA} = \{PA_1, ..., PA_M\}$, global resource capacity profile $[C_{kt}]_{K\times T}$

Output: An allocation $O^G = \{O_1^G, ..., O_M^G\}$

```
Initialization: US ← PA, Ψ ← [C_kt]_{K×T};
while US ≠ ∅ do
    ξ ← 0, γ_max ← 0;
    foreach PA_i ∈ US do
        Ask PA_i a capacity query with profile Ψ ;
        Collect answer B_i = ⟨Λ_i, v_i(Λ_i)⟩;
        Calculate the average price value γ(B_i);
        if γ(B_i) > γ_max then
            ξ ← i, γ_max ← γ(B_i);
        end
    end
    Grant B_ξ to PA_ξ, calculate payment p_ξ, and update item capacity profile
    Ψ ← Ψ − Λ_ξ;
    O_ξ^G ← Λ_ξ, US ← US\{PA_ξ};
end
```

pricing scheme. In the project scheduling literature, usually the cost for utilizing resource r is calculated based on a fixed unit price, e.g. (Beşikci et al., 2015; Naber and Kolisch, 2014; Mao et al., 2009). Under this scheme, the total payment can be determined according to the total utilizing amount of each resource. Here, in the DRCMPSP auction, we denote the unit price of utilizing global resource r_k as g_k. Then the payment p_i for PA_i of obtaining a multiset Λ is:

$$p_i(\Lambda) = \sum_{k=1}^{K} g_k \sum_{t=1}^{T} \lambda(\Lambda, k, t). \tag{3.11}$$

Next, we show a conclusion regarding the fixed unit pricing scheme:

Lemma 3.3 *Under fixed unit pricing scheme, a bidder PA_i in Algorithm 1 only needs to consider submitting bids for cores when answering capacity queries.*

Proof. Given a capacity query with profile Ψ, for any feasible multiset $\Lambda \in \boldsymbol{\Lambda}(\Psi)$, we can always find a core $\widehat{\Lambda} = \widehat{\Lambda}_i(S_i^*(\Lambda))$ with $v_i(\Lambda) = v_i(\widehat{\Lambda})$. It is easy to see

that $\lambda(\widehat{\Lambda}, k, t) \leq \lambda(\Lambda, k, t), \forall k, t$, hence we have $p_i(\widehat{\Lambda}) \leq p_i(\Lambda)$ according to Equation (3.11). Therefore, we have $u_i(\widehat{\Lambda}) \geq u_i(\Lambda)$. In addition, for the two corresponding bids $B_i^1 =< \widehat{\Lambda}, v_i(\widehat{\Lambda}) >$ and $B_i^2 =< \Lambda, v_i(\Lambda) >$, we have $\gamma(B_i^1) \geq \gamma(B_i^2)$. Therefore, whenever Λ is granted, $\widehat{\Lambda}$ can also be granted with an equal or higher utility. □

Lemma 3.3 shows that it is sufficient for PAs to bid for cores, which requires the minimum amount of global resource. In addition, from the first statement of Lemma 3.1 we can conclude that the total number of required units in a bid from PA_i is RQ_i. We can show that under reasonable assumption, the worst-case approximation ratio of the social welfare in (Gonen and Lehmann, 2000) still holds, when the capacity queries can be answered optimally. To be specific, the assumption is that $RQ_i \geq M$ holds for all PA_i, meaning that the number of total requested item units by each bidder should be larger than or equal to the number of bidders. This is a justifiable assumption in DRCMPSP auctions, because the number of demanded resource units for each bidder is usually much larger than the number of bidders in a DRCMPSP case. Below we prove this conclusion based on the proof in (Gonen and Lehmann, 2000).

Theorem 3.2 *Denote the allocation found by Algorithm 1 as O^G. When all capacity queries can be answered optimally, if $\forall i, RQ_i \geq M$, then O^G approximates the optimal allocation O^* within a factor of $\sqrt{H}$.*

Proof. According to Lemma 3.3, a bidder will only submit bids for cores, hence each element O_i^G of O^G should be a core of PA_i, and $\sum_{k=1}^{K}\sum_{t=1}^{T}\lambda(O_i^G, k, t) = RQ_i$. Then the condition becomes $\forall i, M \leq \sum_{k=1}^{K}\sum_{t=1}^{T}\lambda(O_i^G, k, t)$. According to (Gonen and Lehmann, 2000), we can assume that O^G and O^* have no element in common, i.e. $\forall i$, $O_i^G \neq O_i^*$. This is reasonable because if there is any common element, we can remove it from both O^G and O^* along with the items required by it and work on the remaining problem, which can still make the theorem hold. Let $\alpha = SW(O^*) = \sum_{i=1}^{M} v_i(O_i^*)$ and $\beta = SW(O^G) = \sum_{i=1}^{M} v_i(O_i^G)$. We need to show that

$$\alpha \leq \beta\sqrt{K}. \tag{3.12}$$

According to (Gonen and Lehmann, 2000), to prove the above inequality, we only need to show that

$$\sum_{i=1}^{M}\gamma(B_i^*)^2 \leq \sum_{i=1}^{M}\gamma(B_i^G)^2\sum_{k=1}^{K}\sum_{t=1}^{T}\lambda(O_i^G, k, t), \tag{3.13}$$

where $B_i^* = \langle O_i^*, v_i(O_i^*)\rangle$ and $B_i^G = \langle O_i^G, v_i(O_i^G)\rangle$. Let $\boldsymbol{B^*} = \{B_1^*, ..., B_M^*\}$ and $\boldsymbol{B^G} = \{B_1^G, ..., B_M^G\}$. Since O^* and O^G have no element in common, all multisets O_i^* are excluded from O^G, because one of the following two scenarios happens.

- Shading: in O^G, bidder i itself is granted a bid with equal or higher average price, i.e. $\gamma(B_i^*) \leq \gamma(B_i^G)$.
- Conflicting (Gonen and Lehmann, 2000): B_i^* conflicts with some bids in $\boldsymbol{B^G}$ that have already been granted. In other words, in O^G, there are some other bidders being granted bids with equal or higher average prices, and this makes some items I_{kt} requested by B_i^* not enough to be granted.

It should be noted that the above two scenarios cannot happen at the same time for a bid B_i^*. This is because a bidder will be granted only one bid, hence it cannot have conflicts between two bids made by itself. Then, for each bid $B_j^G \in \boldsymbol{B^G}$, there could be a set of bids $B_i^* \in \boldsymbol{B^*}$ for O_i^*, $i \neq j$ that conflict with B_j^G (denote the set as OP_j), and at most one bid B_i^* for O_j^* shaded by B_j^G. Hence $\forall j$, we have

$$\begin{aligned}\sum_{i \in OP_j} \gamma(B_i^*)^2 + \gamma(B_j^*)^2 &\leq (M-1)\gamma(B_j^G)^2 + \gamma(B_j^G)^2 \\ &= M \cdot \gamma(B_j^G)^2 \leq \gamma(B_j^G)^2 \sum_{k=1}^{K}\sum_{t=1}^{T} \lambda(O_i^G, k, t).\end{aligned} \tag{3.14}$$

Therefore, the inequality (3.13) holds, and the proof is complete. □

As stated before, it is NP-hard for a bidder to solve the bidding problems optimally. When approximate answers for capacity queries are used, i.e. the bids $B_i = \langle \Lambda_i, \widetilde{v}_i(\Lambda_i)\rangle$ collected in Line 6 of Algorithm 1 are approximate ones, the theoretical guarantee in Theorem 3.2 does not hold. Note that here we assume that for PA_i, the value of a multiset Λ can only be obtained by finding a feasible schedule $S_i \in \boldsymbol{S}_i(\Lambda)$. This is reasonable because the revenue can only be obtained by completing the whole project, which inevitably requires a feasible schedule. Then, when a capacity query is answered approximately by PA_i with a bid $B_i = \langle \Lambda, \widetilde{v}_i(\Lambda)\rangle$ where $\Lambda = \widehat{\Lambda}_i(S_i)$ is a core generated from a feasible schedule S_i and $\widetilde{v}_i(\Lambda)$ is computed using Equation (3.9), PA_i will obtain a value of $\widetilde{v}_i(\Lambda)$ from Λ, instead of its exact value $v_i(\Lambda)$. Therefore, suppose an allocation O is found, then the social welfare $\widetilde{SW}(O)$ is an approximate one of the actual value $SW(O)$,

and $SW(O) - \widetilde{SW}(O) = \sum_{i=1}^{M} (v_i(O_i) - \widetilde{v}_i(O_i))$. In general, optimal answers to the capacity queries can help to improve the solution quality, and make the theoretical bound in Theorem 3.2 hold. However, for large-size projects, approximate solutions will be more realistic regarding the computational efficiency. When this approach is deployed in reality, a practical way to handle this contradiction is that the auctioneer specifies a time limit T_{ans} for all bidders to compute an answer for a capacity query. Then, a bidder can make use of this time period to conduct *deliberation*, i.e. to keep improving their answers. One way of deliberating is to run anytime algorithms until the time limit is reached.

3.3.3 Improving the Greedy Allocation Using Branch-and-Bound

Algorithm 1 allocates resources to the bidders one by one, which forms a sequence of PAs determined according to the average price function. However, due to the greedy nature of Algorithm 1, some other sequences that could produce allocations with higher social welfare may be ignored. Based on the branch-and-bound algorithm in (Gonen and Lehmann, 2000) designed to optimally solve the multi-unit WDP, here we introduce another strategy, which is a branch-and-bound heuristic that works with capacity queries to find better sequences of PAs, so as to improve the results of Algorithm 1. Similar to Algorithm 1, a bidder in the branch-and-bound heuristic will be granted only one of its bids, such that the substitutability of the valuation functions can be handled.

The branch-and-bound heuristic tries to perform depth-first search on a search tree, which will be constructed during searching. The root node represents an empty sequence, while the interior and leaf nodes represent partial and full sequences, respectively. The searching process starts at the root node. At the root node and each interior node, a function BnB illustrated in Algorithm 2 is called, which requires several parameters: the best allocation value Obj^* found so far and the corresponding sequence Seq^*; the current partial sequence on this node $crtSeq$ and the corresponding value $crtObj$, the set of unallocated bidders $\overline{US}$, and the current resource capacity profile Ψ_0. At each interior node, each bidder in $\overline{US}$ will be asked a capacity query. The bounding operation is conducted by computing $h = \sum_{B_i \in \boldsymbol{B}} v_i(\Lambda_i)$, which

Algorithm 2: BnB(Obj^*, Seq^*, $crtObj$, $crtSeq$, $\overline{US}$, Ψ_0)

Input: The best objective value Obj^*, the best sequence Seq^*, the objective value on the current node $crtObj$, the partial sequence on the current node $crtSeq$, unallocated bidders $\overline{US}$, the resource capacity profile on the current node Ψ_0

if $crtObj > Obj^*$ **then**
  $Obj^* \leftarrow crtObj$, $Seq^* \leftarrow crtSeq$;
end
$\boldsymbol{B} = \emptyset$, $\Psi = \Psi_0$;
foreach $PA_i \in \overline{US}$ **do**
  Ask PA_i a capacity query with profile Ψ, and collect answer B_i,
  $\boldsymbol{B} \leftarrow \boldsymbol{B} \bigcup \{B_i\}$;
end
Compute upper bound $h = \sum_{B_i \in \boldsymbol{B}} v_i(\Lambda_i)$;
if $(crtObj + h) \leq Obj^*$ **then**
  return;
end
Sort $\boldsymbol{B}$ in descending order based on average price;
Create a queue $\boldsymbol{Q}$ of PAs in $\overline{US}$ according to their order in sorted $\boldsymbol{B}$;
while $\boldsymbol{Q} \neq \emptyset$ **do**
  Get the first element PA_ξ in $\boldsymbol{Q}$, and remove it from $\boldsymbol{Q}$;
  Add PA_ξ to $crtSeq$, $crtObj \leftarrow crtObj + v_\xi(\Lambda_\xi)$, $\Psi \leftarrow \Psi - \Lambda_\xi$,
  $\overline{US} \leftarrow \overline{US} \backslash \{PA_\xi\}$;
  BnB(Obj^*, Seq^*, $crtObj$, $crtSeq$, $\overline{US}$, Ψ);
  $crtObj \leftarrow crtObj - v_\xi(\Lambda_\xi)$, $\overline{US} \leftarrow \overline{US} \bigcup \{PA_\xi\}$, $\Psi \leftarrow \Psi + \Lambda_\xi$;
end
return

represents an upper bound of the total value the remaining bidders can contribute to the current partial allocation. The branching operation is conducted according to the average price value, i.e. bidders with higher average price will be added to the partial sequence first. It should be noted that the upper bound h is admissible only if the bidding problem can be answered optimally. When an approximation algorithm is used to solve the bidding problem, some search nodes that can lead to optimal sequences may be deleted, which could misguide the algorithm to return a suboptimal sequence. Therefore, in general Algorithm 2 is a heuristic, though it can guarantee improvement over the results of Algorithm 1. Also note that Algorithm 2 does not guarantee finding the optimal allocation even when the upper bound is admissible. This is because even when the optimal sequence is found, the valuation function of all bidders may not be elicited completely.

For M bidders, the number of all possible allocation sequences is $M!$, which grows extremely fast with M. To control the execution time, we introduce a parameter Q_{nv} in the branch-and-bound heuristic. When the number of nodes visited since the last update of Obj^* reaches Q_{nv}, the algorithm terminates with the best allocation found so far. When $Q_{nv} = 1$, the allocation O^B is the same as O^G produced by the greedy allocation strategy.

In Algorithm 2, a PA may be requested to backtrack such that a better allocation can be found. However, under the fixed unit pricing scheme, backtracking may decrease the utility of a PA, if it is granted a multiset with lower value. To guarantee that the utilities of the PAs will not decrease during the branch-and-bound process, here we adopt the *VCG-based payment* proposed in (Nisan and Ronen, 2007), which is built on top of the well-known VCG (Vickrey (Vickrey, 1961), Clarke (Clarke, 1971), Groves (Groves, 1973)) payment scheme. Usually, VCG payment scheme is used for designing incentive compatible mechanisms in various multi-agent resource allocation applications. When a mechanism (e.g. auction) is incentive compatible, it is in the best interest of the participating agents (e.g. bidders) to reveal their true values to the center (e.g. auctioneer). Different from the typical use of VCG, here we adopt one of its variations, the VCG-based payment, to guarantee that the utility of a PA will not be affected negatively by backtracking. Note that here we still assume that the bidders are willing to reveal their true values in their bids.

VCG-based payment scheme adopts the formulation of VCG payment, and the

only difference is to replace the optimal allocation with a suboptimal one, since the auctioneer may not be able to find the optimal allocation due to the computational complexity. Mathematically, denote the allocation found by the branch-and-bound heuristic as $O^B = \{O_1^B, ..., O_M^B\}$, then the payment for each bidder is:

$$p_i(O_i^B) = -\left(\sum_{j=1, j\neq i}^{M} v_j(O_j^B) - q_i\right), \tag{3.15}$$

where q_i is an arbitrary value calculated without the valuation of bidder i. Then the utility of a bidder when allocated O_i^B can be written as:

$$u_i(O_i^B) = v_i(O_i^B) - p_i(O_i^B) = \sum_{i=1}^{M} v_i(O_i^B) - q_i = SW(O^B) - q_i. \tag{3.16}$$

When approximate answers for capacity queries are used, $SW(O^B)$ in the above equation should be replaced by the approximate one $\widetilde{SW}(O^B)$. Equation (3.16) shows an important property of VCG-based payment, i.e. the utility of a bidder is aligned with the (approximate) social welfare of the allocation produced by the branch-and-bound heuristic. Since the allocation will only be updated when another one with a higher (approximate) social welfare is found (Lines 1-3 of Algorithm 2), the utility of a bidder will not decrease in the branch-and-bound process.

Below we show a conclusion similar to Lemma 3.3:

Proposition 3.2 *Under VCG-based payment scheme, a bidder PA_i in the branch-and-bound heuristic only needs to consider submitting bids for cores when answering capacity queries.*

Proof. Similar to the proof of Lemma 3.3, given a capacity query with profile Ψ, for any feasible multiset $\Lambda \in \mathbf{\Lambda}(\Psi)$, we can always find a core $\widehat{\Lambda} = \widehat{\Lambda}_i(S_i^*(\Lambda))$ with $v_i(\Lambda) = v_i(\widehat{\Lambda})$, and $\lambda(\widehat{\Lambda}, k, t) \leq \lambda(\Lambda, k, t), \forall k, t$. Then, for any allocation O with $O_i = \Lambda$, there always exists an allocation $\widehat{O}$ with $\widehat{O}_i = \widehat{\Lambda}$ such that $SW(\widehat{O}) \geq SW(O)$. This is because the additionally required items in Λ could potentially block bids from other bidders which could help in producing a better allocation. Therefore, PA_i can gain a equal or larger utility by bidding on $\widehat{\Lambda}$ than Λ. □

One way to compute q_i in Equation (3.15) is to adopt the formulation of the Clarke mechanism (Clarke, 1971). More specifically, for each bidder i, run the branch-and-bound heuristic without the participation of bidder i, and denote the

allocation found as $O^{B,i} = \{O_1^{B,i}, ..., O_M^{B,i}\}$, where $O_i^{B,i} = \mathbf{0}$. Then, set $q_i = \sum_{j=1}^{M} v_j(O_j^{B,i})$. Due to the heuristic nature of the algorithm, it is possible that the allocation of some $O^{B,i}$ may have higher social welfare than O^B (Nisan and Ronen, 2007). Therefore, the allocation O^B in Equation (3.16) should be replaced by $O^{B*} = \underset{O \in \boldsymbol{O^B}}{\operatorname{argmax}} \sum_{i=1}^{M} v_i(O_i)$, where $\boldsymbol{O^B} = \{O^B, O^{B,1}, ..., O^{B,M}\}$, such that an equal or higher social welfare (and higher utilities for PAs) can be achieved.

3.4 Empirical Evaluation

In this section, we conduct experiments on some benchmark problems to verify the effectiveness of our approach. First, we compare the greedy allocation strategy against state-of-the-art approaches. Then, we examine the improvements the branch-and-bound strategy can make to the greedy allocation. Since all previous approaches aim at minimizing APD, to make a fair comparison, here we set the delay cost for all bidders to be linear with the same unit cost w_i, and evaluate the solution quality according to the APD values.

As mentioned in Section 3.3.2, a reasonable choice for a bidder to answer the capacity queries is to using anytime algorithms to keep improving the answers until the time limit is reached. However, since the benchmark cases consist of large projects (up to 120 activities per project), it may take quite a long time for a bidder to find a feasible solution for a bidding problem. Here, in the experiments, to make the computation scalable, we design a fast priority-rule based scheduling algorithm (Algorithm 7) to approximately solve the bidding problem in polynomial-time. Details of Algorithm 7 can be found in the Appendix. Algorithm 7 will be used in most of the following experiments to simulate the process of answering capacity queries. We also examine the impact of deliberation in Section 3.4.1.1, by solving the bidding problem using anytime solvers with certain deliberation time T_{ans}.

3.4.1 Results of the Greedy Allocation Strategy

We test the greedy allocation strategy on two problem sets. The first one contains all the 140 cases from MPSPLIB.[5] To the best of our knowledge, this is the only public benchmark of DRCMPSP. We compare our approach with state-of-the-art

[5]http://www.mpsplib.com/

Table 3.1: Comparison of Average APD Values with Other Approaches

Subset	Greedy	DMAS/EM	DMAS/ABN	SASP
MP_30_2	13.60	**8.90**	15.90	22.40
MP_90_2	**5.80**	6.60	9.90	18.50
MP_120_2	**50.70**	59.40	67.20	69.10
MP_30_5	19.08	**17.00**	21.20	31.90
MP_90_5	11.20	**4.60**	11.00	23.80
MP_120_5	**46.00**	54.20	66.48	71.90
MP_30_10	**55.92**	66.40	87.50	90.20
MP_90_10	**38.14**	50.90	46.08	65.00
MP_120_10	**107.58**	119.60	130.96	139.60
MP_30_20	**116.23**	138.00	207.96	185.50
MP_90_20	**21.28**	27.40	30.22	48.60
MP_120_20	**24.22**	28.60	37.18	61.10
MP_90_2AC	**108.35**	126.00	144.15	158.60
MP_120_2AC	**38.30**	52.40	47.00	56.70
MP_90_5AC	**249.72**	284.60	384.08	404.60
MP_120_5AC	**181.76**	233.50	291.44	258.80
MP_90_10AC	**175.91**	223.30	313.33	283.90
MP_120_10AC	**104.31**	169.40	171.54	181.00
MP_90_20AC	**94.97**	126.70	146.37	161.80
MP_120_20AC	**163.54**	280.70	297.39	297.40

distributed approaches DMAS/EM (Zheng et al., 2014), DMAS/ABN (Adhau et al., 2012), and a centralized approach SASP on this problem set. We also compare our approach with another combinatorial auction based approach in (Confessore et al., 2007) (denoted as Confessore's). Since this approach can only handle one single-unit global resource, we conduct experiments on the second problem set, which is generated from MPSPLIB by replacing the multi-unit global resources in each case with one single-unit global resource.

3.4.1.1 Experiments on the First Problem Set

The 140 cases from MPSPLIB are divided into 20 subsets named as "MP_N_M", where $N \in\{30, 90, 120\}$ is the number of activities per project, and $M \in\{2, 5, 10, 20\}$ is the number of projects. Thus the largest cases contain $20 \times 120 = 2400$ activities. Each case contains 4 resources per project, and the number of global resources K in each case is chosen between 1 and 4. The cases with no local resource, i.e. $K = 4$, are called "Agent Cooperation" cases, and a postfix "AC" is added to the subset names. Each AC subset contains 10 cases, while each non-AC subset contains 5 cases. To measure the tightness of global resource constraints, a Utilization Factor (UF) (Homberger, 2012) value is calculated for each case. Here we briefly introduce how to compute UF. For a DRCMPSP case, a utilization factor UF_k for each global resource r_k is computed as follows:

$$UF_k = \frac{\sum_{i=1}^{M} \sum_{j=1}^{N_i} b_{ij}^k}{C_k \cdot GCPL}, \tag{3.17}$$

where C_k is the average capacity of r_k per time slot, and $GCPL$ is the global critical path length of all projects. Then, UF is set to be the maximum UF_k of all the global resources, i.e. $UF = \max \{UF_k | k \in \{1, .., K\}$. $UF < 1$ indicates a low to medium resource constraint, while $UF > 1$ indicates a medium to high constraint (Lova and Tormos, 2001). Average UF value $\overline{UF}$ of each subset is listed in Table 3.2. In general, AC subsets have higher $\overline{UF}$ than non-AC subsets given the same M and N, which makes AC subsets harder to solve.

We implement Algorithm 1 using Java 1.8, and run the algorithm on a single Intel Xeon Workstation (3.5GHz, 16GB). We set the scheduling horizon $T = 1500$ for all the cases. We first use Algorithm 7 to solve the bidding problems. All the 140 cases are successfully solved within a total time of 140 seconds. We calculate the

Table 3.2: Average Utilization Factor of Each Problem Subset

Subset	$\overline{UF}$	Subset	$\overline{UF}$	Subset	$\overline{UF}$	Subset	$\overline{UF}$
MP30_2	0.84	MP120_5	1.32	MP90_20	0.90	MP120_5AC	3.80
MP90_2	0.57	MP30_10	2.38	MP120_20	0.87	MP90_10AC	3.85
MP120_2	1.31	MP90_10	1.14	MP90_2AC	2.27	MP120_10AC	2.61
MP30_5	0.82	MP120_10	1.91	MP120_2AC	1.36	MP90_20AC	2.70
MP90_5	0.61	MP30_20	3.37	MP90_5AC	4.99	MP120_20AC	3.65

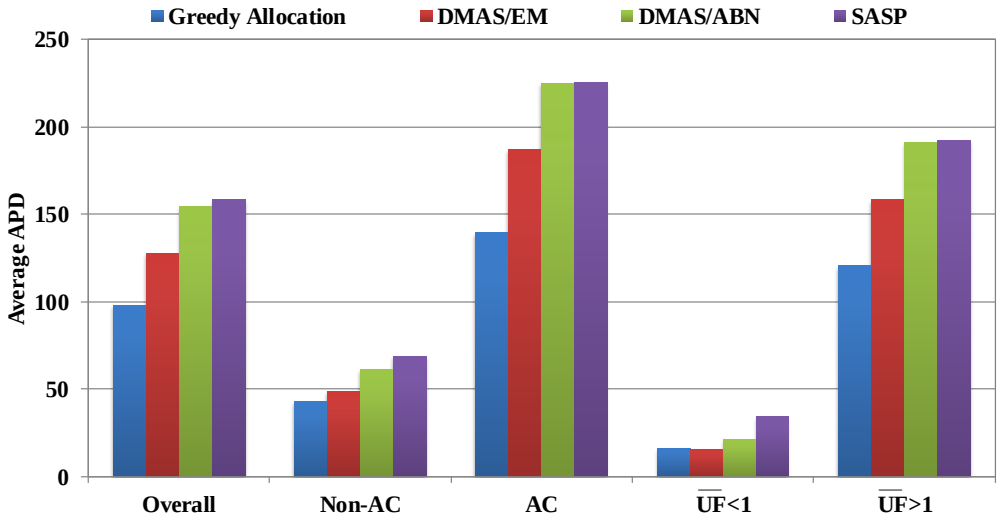


Figure 3.2: Comparison of Average APD

average APD of each subset as shown in Table 3.1, where the best results are marked as bold. Among the 20 subsets, the greedy allocation strategy outperforms other approaches in 17 subsets. For MP_120_20AC, one of the most complex subset with the tightest resource constraints, our approach outperforms DMAS/EM by 41.7%. According to Equation (3.1), the minimum possible APD value is 0. We observe that for 6 cases, the greedy allocation strategy finds the solutions with $APD = 0$, which shows that it has the ability to find the optimal solution and confirms the correctness of our combinatorial auction formulation of DRCMPSP.

To evaluate the performance of the greedy allocation strategy in different types of subsets, we first split the problem subsets into two groups that consist of only non-AC and AC cases, and calculate the average APD obtained by the four approaches. Then we group the subsets according to if $\overline{UF} > 1$, and calculate the corresponding

Table 3.3: Comparison with DMAS/EM

Type	Overall	Non-AC	AC	$\overline{UF} < 1$	$\overline{UF} > 1$
Greedy Allocation	**97.98**	**42.48**	**139.61**	15.86	**120.38**
DMAS/EM	129.89	48.47	187.08	**15.52**	158.26
Difference (%)	-24.57	-12.36	-25.37	2.19	-23.94

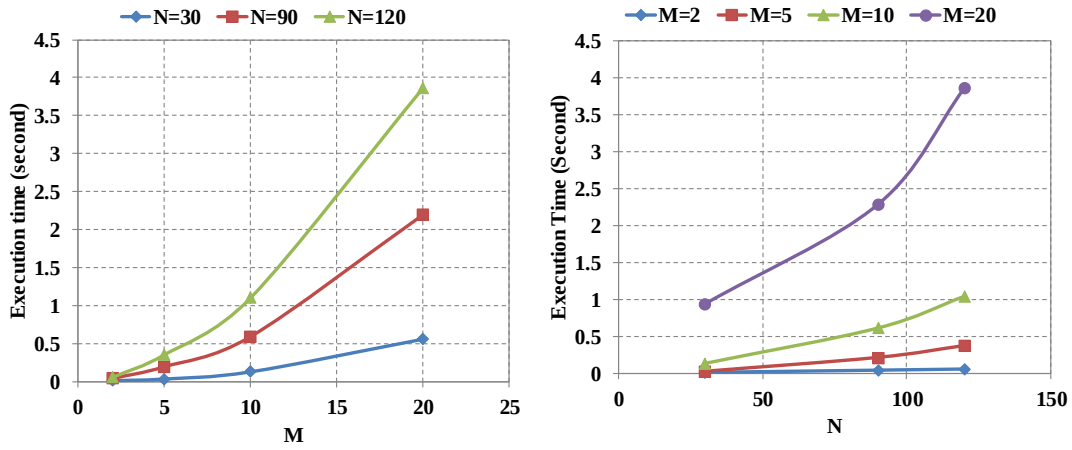


Figure 3.3: Execution Time on MPSPLIB Cases: (left) fix N^*, and (right) fix M

average APD. We plot these values along with the overall average APD of all the 140 cases in Figure 3.2. As presented, our greedy allocation strategy produces the lowest average APD compared with other three approaches. Moreover, our greedy allocation strategy generates better results for three types of subsets non-AC, AC, and $\overline{UF} > 1$; for $\overline{UF} < 1$, our result is comparable with the best result obtained by DMAS/EM. In Table 3.3, we compare our results with DMAS/EM, which produces the closest results to ours. As shown in Table 3.3, overall, our approach outperforms DMAS/EM by 24.57%. For the two harder groups AC and $\overline{UF} > 1$, our approach gives 25.37% and 23.94% improvements against DMAS/EM, respectively. In general, our greedy allocation approach gives lower APD compared to other three approaches, and performs better on harder subsets with tighter global resource constraints.

We then evaluate the scalability of the greedy allocation strategy using Algorithm 7 for solving bidding problem according to the total time for solving one case, since it is simulated on a single computer. Before showing the results, we first give a brief analysis on the complexity of Algorithm 1. Apparently, the number of iterations

between Line 2 and 9 is M, since in each round a bidder will be allocated and removed from $\overline{US}$. Hence, the worst case for a bidder is to generate M bids, and the total number of bids generated by all bidders is $M(M+1)/2$. If Algorithm 7 is used, the worst-case complexity of a bidder is $O\left(MN_i^2(K+L_i)d_i^*\right)$. The main task for the auctioneer in each round is to calculate the average price and maintain the capacity profile, leading to a worst-case complexity of $O(MKT)$. Since d_i^* and $K+L_i$ are constants in all the cases of MPSPLIB and T is fixed, the worst-case complexity of the MA and PA are $O(M)$ and $O(MN_i^2)$. Thus, if Algorithm 1 is simulated on a single thread, the worst-case complexity to generate a solution is $O(M^2N^{*2})$, where $N^* = \max\{N_1, ..., N_M\}$.

We plot the average execution time against M and N^* in Figure 3.3, by fixing one parameter and increasing the other. The time increasing trends in Figure 3.3 are compatible with our complexity analysis and show that the greedy allocation strategy with polynomial-time bidding algorithm can efficiently solve large problem cases with thousands of activities from tens of projects.

To evaluate the impact of the bidder's deliberation, we simulate the capacity answering using standard solver. Specifically, we model the RCPSP as an Integer Linear Program (ILP) and solve it using the Gurobi solver (Gurobi Optimization, 2015). To control the deliberation time, we set the time limit T_{ans} of the solver to be 10, 20, 30, 40, 50 seconds. To guarantee that a feasible solution will be found, we first use Algorithm 7 to generate a feasible solution, then use the solver to improve the result until the time limit is reached. We only test the cases with $M \in \{2, 5\}$, since the approach is executed using one thread and the execution time increases quadratically with M, according to the complexity analysis in the previous paragraph[6]. Totally, we test 70 cases out of the 140 cases from MPSPLIB. We list the results according to T_{ans} in Table 3.4, where the column ALG3 consists of the results of the corresponding subset from Table 3.1. As shown in this table, in general, a longer deliberation time tends to produce better solutions. Compared with the results produced with Algorithm 7, we can conclude that for all the Non-AC subsets, the deliberation helps in improving the results of Algorithm 7, while the solution quality could drop for the AC subsets. An explanation for this observation is that, the deliberation enables a bidder to find a better schedule which utilizes

[6]However, in reality the bidders should deliberate in parallel.

Table 3.4: Average APD Values of Greedy Allocation with Different Deliberation Time

Subset	ALG3	T_{ans}=10	T_{ans}=20	T_{ans}=30	T_{ans}=40	T_{ans}=50
MP30_2	13.60	12.90	12.40	12.30	11.90	11.90
MP90_2	5.80	5.80	5.40	5.40	5.40	5.40
MP120_2	50.70	50.10	49.90	49.90	49.20	48.00
MP30_5	19.08	18.00	17.24	16.60	16.40	16.36
MP90_5	11.20	9.84	9.84	9.84	9.84	9.84
MP120_5	46.00	45.16	45.12	44.84	44.96	44.96
MP90_2AC	108.35	109.50	109.50	109.50	109.50	109.50
MP120_2AC	38.30	37.95	37.95	37.95	37.95	37.95
MP90_5AC	249.72	252.10	251.96	251.80	251.80	251.80
MP120_5AC	181.76	182.74	182.68	182.68	182.68	182.68

its own local resources more effectively. In contrast, when all resources are shared, better schedules of the bidders who are allocated earlier may affect the utilities of the bidders who will be allocated later.

3.4.1.2 Experiments on the Second Problem Set

In this section, we describe the experiments on the second problem set, which contains 140 cases generated from the MPSPLIB. More specifically, for each case in MPSPLIB, we first keep the global resource that has the minimum capacity and remove other ones. Then, we replace the capacity of that resource and requirements of each activity on that resource with 1. We classify the newly generated cases based on the number of activities and projects. Hence, 12 subsets can be obtained.

We implement Confessore's approach using Java 1.8 and run both Confessore's and Algorithm 1 on the same workstation as we explained in the previous section. Algorithm 7 is used here to solve the bidding problem. During the experimentation, we observe that Confessore's approach cannot converge on some cases. Hence, we limit the maximum iterations of this approach to 3000. The scheduling horizon T is set to 15000.

All the 140 cases in this problem set are successfully solved by our approach

Table 3.5: Number of Cases Solved by Confessore's Approach for Each Subset of the Second Problem Set

Subset	Total	Solved	Ratio (%)	Subset	Total	Solved	Ratio (%)
MP30_2	5	5	100	MP30_10	5	4	80
MP90_2	15	15	100	MP90_10	15	10	67
MP120_2	15	15	100	MP120_10	15	14	93
MP30_5	5	5	100	MP30_20	5	4	80
MP90_5	15	12	80	MP90_20	15	3	20
MP120_5	15	14	93	MP120_20	15	9	60

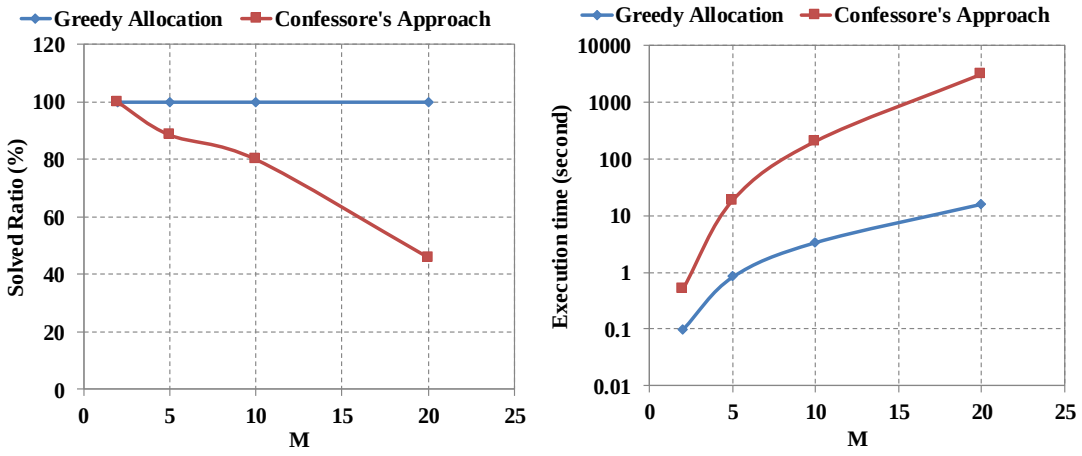


Figure 3.4: Scalability Comparison on the Second Problem Set

within 730 seconds, while Confessore's approach can solve 79% (110/140) of all cases in about 15 hours. We list the number of cases solved by Confessore's approach for each subset in Table 3.5, and plot the ratio of solved cases against the number of projects M in the left of Figure 3.4. The trend in this figure indicates that Confessore's approach may not be able to solve large cases where tens of projects are involved. To compare the computational efficiency, we calculate the average execution time of the cases successfully solved by both approaches regarding the number of projects M, and plot the curves in the right of Figure 3.4 (the vertical axes is in log scale). As shown in this figure, the execution time of our greedy allocation strategy are much smaller than those of Confessore's approach, and when N is larger than 10, our approach can be two orders of magnitude faster.

Table 3.6: Comparison of Average APD Values on the Second Problem Set

Subset	Greedy	Confessore's	Diff(%)
MP30_2	**106.40**	122.80	-13.36
MP90_2	**354.27**	393.70	-10.02
MP120_2	**362.20**	439.60	-17.61
MP30_5	**167.28**	279.80	-40.21
MP90_5	**641.02**	869.67	-26.29
MP120_5	**993.36**	1281.24	-22.47
MP30_10	**393.30**	667.08	-41.04
MP90_10	**1504.75**	2111.30	-28.73
MP120_10	**1564.11**	2360.66	-33.74
MP30_20	**937.39**	1540.06	-39.13
MP90_20	**1591.27**	2800.17	-43.17
MP120_20	**3037.74**	5075.11	-40.14

We then evaluate the solution quality of these two approaches by calculating the average APD of the cases in each subset in Table 3.6. Here we only consider the cases that are solved by both approaches. From this table, we can conclude that our approach consistently achieves lower APD values on all the subsets. Furthermore, the improvement tends to increase on harder cases having more activities and projects. The results show that our approach is computationally frugal, and can efficiently solve large cases where tens of projects are involved.

3.4.2 Results of the Branch-and-bound Strategy

We implement and run the branch-and-bound heuristic on the 140 cases from MPSPLIB, with the same setting as in Section 3.4.1.1. The algorithm is executed with different Q_{nv} values chosen from $[10, 1000]$. We also use Algorithm 7 to solve the bidding problem. Table 3.7 shows the results for Q_{nv} equals to 10, 100, 500, and 1000. We can observe that for all subsets except those where $M = 2$, solution quality is improved. This makes sense since there are only 2 possible sequences when $M = 2$. Table 3.7 also shows that the improvement over the greedy allocation increases with Q_{nv}. We plot the increase of the improvement (Impr) in Figure 3.5, along with the

Table 3.7: Average APD Values of the Branch-and-bound Strategy with Different Q_{nv}

Subset	Greedy	$Q_{nv} = 10$	$Q_{nv} = 100$	$Q_{nv} = 500$	$Q_{nv} = 1000$
MP30_2	13.60	13.00	13.00	13.00	13.00
MP90_2	5.80	5.60	5.60	5.60	5.60
MP120_2	50.70	50.70	50.70	50.70	50.70
MP30_5	19.08	17.48	17.28	17.28	17.28
MP90_5	11.20	9.04	9.04	9.04	9.04
MP120_5	46.00	45.36	44.60	44.60	44.60
MP30_10	55.92	55.34	54.72	53.40	53.12
MP90_10	38.14	36.72	36.26	35.72	35.58
MP120_10	107.58	106.38	105.74	104.84	103.84
MP30_20	116.23	115.70	115.50	115.46	115.37
MP90_20	21.28	20.82	20.25	20.25	20.24
MP120_20	24.22	24.09	23.84	23.43	23.42
MP90_2AC	108.35	107.20	107.20	107.20	107.20
MP120_2AC	38.30	37.80	37.80	37.80	37.80
MP90_5AC	249.72	248.62	247.28	247.28	247.28
MP120_5AC	181.76	178.64	176.88	176.88	176.88
MP90_10AC	175.91	175.52	175.43	174.76	174.64
MP120_10AC	104.31	103.71	103.10	102.87	102.63
MP90_20AC	94.97	94.85	94.47	94.27	94.04
MP120_20AC	163.54	163.14	162.95	162.72	162.64
Average APD	97.98	97.11	96.67	96.46	96.36
Improvement (%)	N/A	0.88	1.33	1.55	1.65

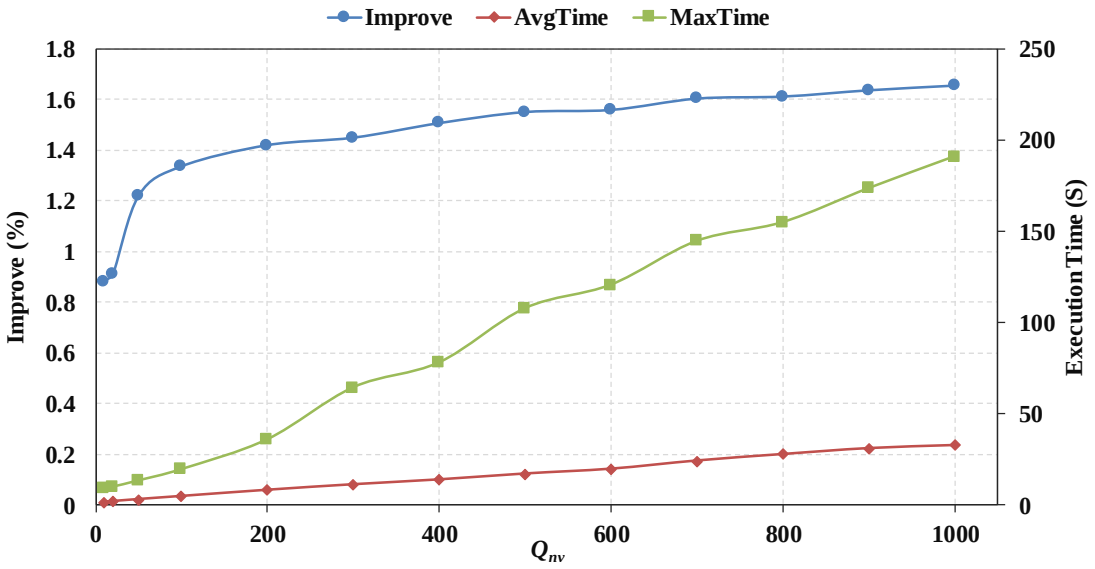


Figure 3.5: Improvement and Execution Time of the Branch-and-bound Strategy

average execution time per case (AvgTime) and the maximum execution time for a case (MaxTime). When $Q_{nv} = 1000$, the maximum execution time for a case is 191s, which is acceptable since we are dealing with an offline static scheduling problem. Here we only run the algorithm to find the allocation O^B without the computation of VCG-based payment. However, since the sizes of problems of finding O_i^B for computing payments for each bidder is smaller than the problem of finding O^B, the increasing trend of the execution time should be in accordance with the curves in Figure 3.5. To summarize, the branch-and-bound strategy can effectively improve the solution quality of the greedy allocation strategy, within reasonable computation time.

3.5 Conclusions

In this chapter, we study how to solve DRCMPSP using the multi-unit combinatorial auction, since it naturally suits the requirement of DRCMPSP, i.e. making resource allocation decisions without the need of knowing private project information. To this end, we make three contributions. Firstly, we formally analyze the relations between the optimal allocations of the auction and the optimal solutions of the DRCMPSP case. Secondly, to deal with the hard valuation problem in the formulation, we introduce the capacity query to elicit the valuations of the bidders in the DRCMPSP

auction. Thirdly, we employ two strategies proposed in (Gonen and Lehmann, 2000) that work with the capacity queries to efficiently find good allocations for the DRCMPSP auction.

The first strategy allocates the global resources in a greedy manner, which employs an iterative process of querying and allocating. In each round, the bidder with a bid that has the maximum average price will be granted. We show that under fixed unit pricing scheme, the bidders will bid only for cores, which are compact resource utilization profiles of their projects. We also show that when the bidders can answer the capacity queries optimally and requires sufficient large amounts of global resources, the greedy allocation strategy is shown to preserve a worst-case approximation guarantee for DRCMPSP auctions, which is the same as the one proved in (Gonen and Lehmann, 2000) for the winner determination problems.

The second strategy is based on a branch-and-bound process, which aims at improving the greedy allocation by finding a better sequence of the bidders for granting their bids. To guarantee that the bidders' utilities will not decrease during backtrakcing, VCG-based payment in (Nisan and Ronen, 2007) is adopted to align the utility of each bidder to the social welfare of the final allocation.

We conduct experiments on benchmark problem sets. The two proposed strategies show good performance both in solution quality and scalability. Specifically, the greedy allocation strategy can produce solutions with higher quality than state-of-the-art approaches, and the improvement on solution quality tends to be higher on harder cases with more projects, more activities and higher resource contention. The computation could be very efficient when polynomial-time algorithms are used by the bidders to find approximate answers to the capacity queries. Also, the branch-and-bound strategy is shown to be effective in further improving the results of the greedy allocation, and can scale to large cases within reasonable computation time.

Chapter 4

Risk-Neutral Proactive Scheduling with Time-dependent Workability Uncertainty

In the previous chapter, we study a scheduling problem in a distributed environment with deterministic activity durations. However, real-world activities often have random durations, since the execution could be affected by various uncertainties. In this chapter, we discuss how to deal with a centralized scheduling problem, but with uncertain activity durations. More specifically, we study the proactive scheduling problem for RCPSP, aiming at minimizing the expected makespan which is a risk-neutral objective. Different from previous research on this topic, we allow the activity duration uncertainty to be time-dependent, caused by the *time-dependent workability uncertainty*. To circumvent the underlying complexity caused by the uncertainty, we propose a principled approximate approach with convergence guarantee based on Sample Average Approximation (SAA). Finally, we design two efficient branch-and-bound algorithms to optimally solve the SAA problem, which is proved to be NP-hard. Experimental results on benchmark problem instances and different models of activity duration uncertainties confirm the effectiveness of our algorithms.

This chapter is organized as follows. Section 4.1 introduces two types of solutions that will be used in our work, i.e. Partial-Order Schedule and AON-Flow Network. In Section 3.2, we describe our generalized activity duration model and formulate our proactive scheduling problem. In Section 4.3, we show how to approximate the

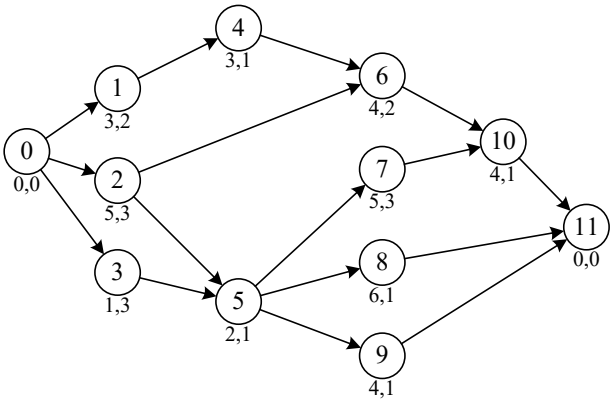


Figure 4.1: An Example of the AON Network

hard proactive scheduling problem using SAA. In Sections 4.4 and 4.5, we present the two proactive scheduling algorithms. Section 4.6 is dedicated to the numerical experiments and analysis. The chapter ends with the conclusions in Section 4.7.

4.1 Preliminaries: POS and AON-Flow Network

In this section, we introduce two concepts that are very useful in handling unforeseen disruptions in project execution, i.e. Partial Order Schedule (POS) and AON-Flow Network. We begin with the graph representation of the temporal constraints.

It is well known that the precedence relations between activities within a project can be represented as an Activity-On-Node (AON) network. Recall that for a RCPSP instance, $A_p = A \cup \{a_0, a_{N+1}\}$ is the set of activities including the dummy ones, and E_p is the precedence relations between them. Then the AON network of this project is $G_p = (A_p, E_p)$, which is a directed acyclic graph (DAG). The AON network of a sample project containing 10 activities is shown in Figure 4.1, where a circle represents an activity, and the left and right numbers below it are the duration and requirement for a single resource with limited capacity of 4 units, respectively. The arrows between circles represent the precedence relations. Throughout this thesis, let $V(G)$ and $E(G)$ be the vertex set and edge set of a graph G, respectively. We also denote $Tr(G)$ as the transitive closure of G, where $(a_i, a_j) \in Tr(G)$ indicates there is a path from a_i to a_j.

When the activity durations are uncertain, a feasible schedule S generated by solving a deterministic instance could be disrupted and become infeasible during

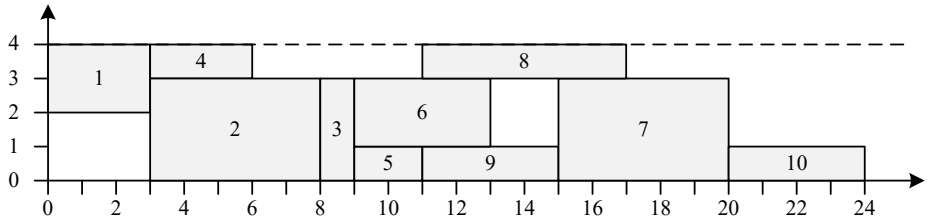


Figure 4.2: A Feasible Schedule

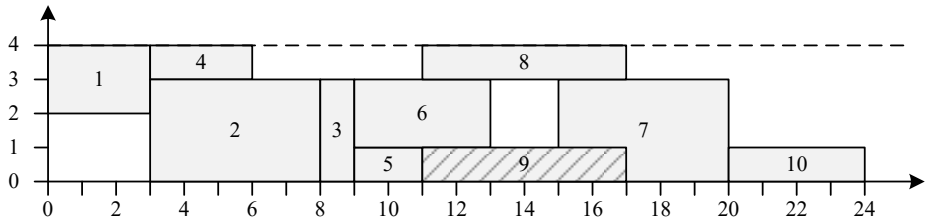


Figure 4.3: An Example of Schedule Disruption

execution, since the activity completion times are also uncertain. Below we give a simple example of disruption. A feasible schedule of the project in Figure 4.1 is shown in Figure 4.2. Suppose that during execution, the duration of activity a_9 is elongated from 4 to 6, as shown in Figure 4.3. This unexpected duration elongation makes the schedule infeasible, since the accumulated resource requirement from time 15 to 17 exceeds the capacity of 4. Whenever such disruption happens, the schedule must be repaired (i.e. restore its feasibility) so that the project execution can be resumed. The repair procedures are often expensive and complex, since both temporal and resource constraints need to be taken into account.

Different from the start-time schedules, POS (Policella et al., 2004) offers a flexible way for project execution, where the start time of each activity is determined during execution time instead of before execution. A POS is a DAG $G_R = (A_p, E_p \cup E_R)$ that augments the AON network G_p by adding an additional set of precedence constraints E_R, such that any temporal feasible solution of G_R is also resource feasible. A sample POS for the project represented in Figure 4.1 is shown in Figure 4.4, where the dotted arrows represent the additional precedence constraints in E_R. The additional temporal constraints in E_R must be chosen carefully, such that G_R is acyclic and all the possible resource conflicts are resolved. We denote the set of POS as $\boldsymbol{G}_R$. In Section 4.4 and 4.5, we will further discuss how to generate a feasible

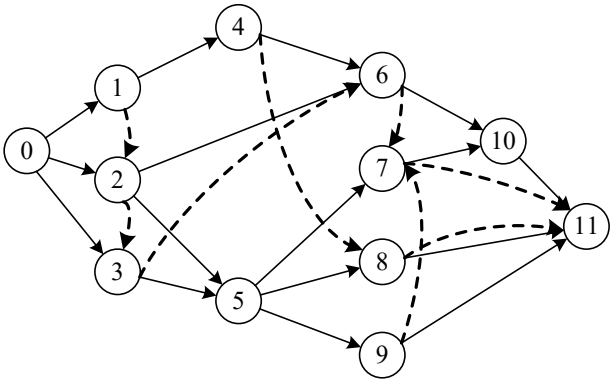


Figure 4.4: An Example of POS

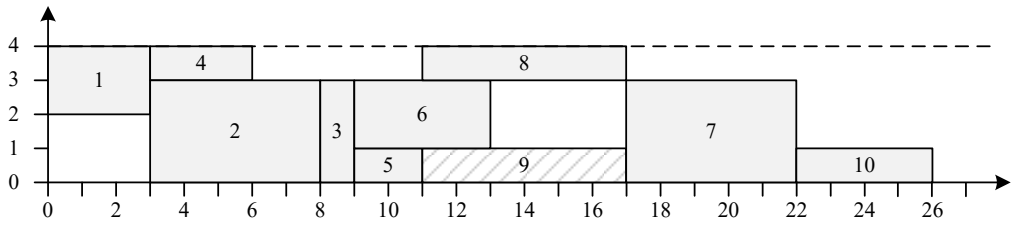


Figure 4.5: The Schedule Obtained by Executing the POS

POS by finding suitable E_R.

POS provides a very efficient way for dealing with uncertain durations, by removing the requirement of reasoning on complex resource constraints. Specifically, the start time of each activity is determined solely based on the precedence constraints and actual activity durations. Given a POS G_R, let $Pre(a_i, G_R) = \{a_j \in A_p | (a_j, a_i) \in E_p \cup E_R\}$ be the immediate predecessors of an activity a_i specified by G_R. During execution time, the start time s_i of a_i can be computed very easily using the equation below:

$$s_i = \max\{c_j = s_j + d_j | a_j \in Pre(a_i, G_R)\}, \tag{4.1}$$

where d_j is the actual duration of a_j. In other words, a_i is started after the completion of all its immediate predecessors in G_R. Along with execution, the start times of all $a_i \in A_p$ can be determined, hence a feasible schedule is obtained. Figure 4.5 shows the actual schedule obtained by executing the project according to the POS in Figure 4.4, with the unexpected elongation of a_9 shown in Figure 4.3. According to the POS, a_7 can only start after a_5, a_6 and a_9 are all completed. Therefore, when the duration of a_9 is elongated to 6, the start time of a_7 will be set to 17,

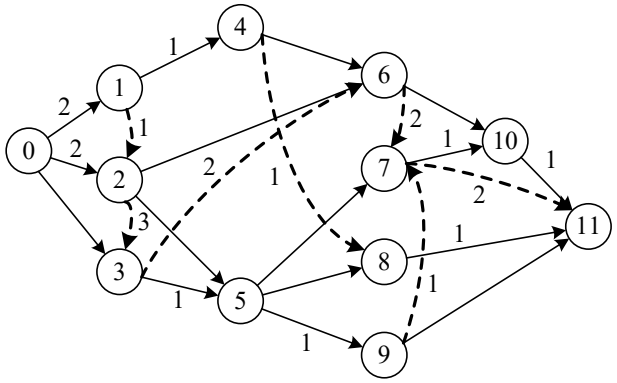


Figure 4.6: An Example of AON-flow Network

which further makes a_{11} to start at 22. Clearly, the schedule shown in Figure 4.5 is conflict-free.

Another concept that will be used here is the AON-flow Network (Artigues et al., 2003). Though POS eliminates the requirements for resource reasoning, the resource allocation decision (i.e. which resource unit is allocated to which activity) is not clearly specified. The resource allocation decision is considered to be very important in practice, since it can provide great advantage in preparing and coordinating the execution process (Leus and Herroelen, 2004). AON-flow Network explicitly specifies such resource allocation decision by specifying *resource flows* between activities. Similar to POS, an AON-flow Network $G_F = (A_p, E_p \cup E_F)$ is also an augmented DAG of the original AON network G_p. The difference lies in the set E_F, where each element $(a_i, a_j) \in E_F$ carries a resource flow from a_i to a_j. The flow is represented as a vector $f_{ij} = (f_{ij1}, ..., f_{ijK})$, where $0 \leq f_{ijk} \leq C_k$ is the amount of resource r_k being transfered from a_i to a_j, i.e. a_i will release f_{ijk} of r_k to a_j after completion. It should be noted that $E_p \cap E_F$ is not necessarily $\emptyset$, which means some edges in the original AON network E_p may also carry resource flows. Figure 4.6 shows a sample AON-flow Network for the project in Figure 4.1, where the arrows associated with values represent the resource carrying edges in E_F, and the values indicate the transfered resource units.

According to (Artigues et al., 2003), the resource flows in a feasible AON-flow Network should satisfy a set of conditions since the problem context is resource-constrained. For convenience, let the requirement for each resource r_k of the two

dummy activities be $b_0^k = b_{N+1}^k = C_k$.[1] Then G_F should satisfy the conditions below:

- Positive flow: $\sum_{r_k \in R} f_{ijk} > 0, \forall (a_i, a_j) \in E_F$;
- Inflow balance: $\sum_{(a_j, a_i) \in E_F} f_{jik} = b_i^k, \forall r_k \in R, a_i \in A_p \setminus \{a_0\}$;
- Outflow balance: $\sum_{(a_i, a_j) \in E_F} f_{ijk} = b_i^k, \forall r_k \in R, a_i \in A_p \setminus \{a_{N+1}\}$.

The positive flow condition guarantees that no edge in E_F carries zero resource flow. The inflow and outflow balance guarantee that the total resource units received and sent by an activity should equal its requirement. We denote the set of AON-flow Networks as $\boldsymbol{G}_F$.

Intuitively, POS and AON-flow Network have close relationship since they share some similarities in their definitions and structures. For example, an AON-flow Network is also a POS since all the possible resource conflicts are resolved by the resource flows. We will further analyze the connections between these two concepts in Section 4.4.1.

4.2 Problem Formulation

In this section, we formulate our proactive scheduling problem. We first generalize the traditional stochastic RCPSP model with the time-dependent workability uncertainty in Section 4.2.1, and then formulate the stochastic optimization model for the proactive scheduling problem in Section 4.2.2.

4.2.1 The Model of Uncertainty

The problem we studied in this chapter shares almost the same statement as the deterministic RCPSP in the earlier part of Section 3.1, expect the duration of an activity a_i is now a random variable $\boldsymbol{D}_i$, instead of a deterministic value. Here we abuse the notation and use d_i to denote the realization of $\boldsymbol{D}_i$. Below we first formulate $\boldsymbol{D}_i$ under the time-dependent workability uncertainty.

Without loss of generality, we first classify all activities in A into Z types, and activities of the same type can be described by the same workability uncertainty model (e.g. requiring the same weather condition). For each activity type z, its

[1] Note that this does not affect the problem, since a_0 and a_{N+1} have zero durations.

(uncertain) workability in a time slot $t \in \{1, ..., T\}$ can be represented by a binary random variable $\boldsymbol{X}_{zt}$, where its realization $x_{zt} \in \{0, 1\}$ and activities of type z can only work on t when $x_{zt} = 1$.[2] Then, the workability model of activity type z can be represented by a random vector $\boldsymbol{X}_z = (\boldsymbol{X}_{z1}, ..., \boldsymbol{X}_{zT})$. Note that here we do not require $\boldsymbol{X}_{zt}$ to be independent of each other. Therefore, depending on applications, $\boldsymbol{X}_z$ can also be described by complex models, e.g. a (truncated) random process. The complete workability uncertainty model can be represented as a random matrix $\boldsymbol{X} = [\boldsymbol{X}_{zt}]_{Z \times T}$, where row z is the random vector for activity type z.

Next, we discuss the probabilistic activity durations under the workability uncertainty. We first assume that this is the only uncertainty source, and each activity a_i has a fixed *baseline duration* d_i^s. Specifically, d_i^s is the condition for determining the completion of an activity: once started, a_i must acquire at least d_i^s workable time slots before completion. Then we can formulate the cumulative distribution function (CDF) of the random duration $\boldsymbol{D}_i$ of an activity a_i, conditioning on its start time s_i as:

$$\begin{aligned} F_{\boldsymbol{D}_i}(d_i \mid s_i = t) &= P(\boldsymbol{D}_i \leq d_i \mid s_i = t) \\ &= P\left(\sum_{\tau=t}^{t+d_i-1} \boldsymbol{X}_{z_i\tau} \geq d_i^s \middle| s_i = t \right), \end{aligned} \tag{4.2}$$

where d_i is the realization of $\boldsymbol{D}_i$, and $z_i \in \{1, ..., Z\}$ is the type of a_i. Clearly, $\boldsymbol{D}_i$ depends on s_i since $\boldsymbol{X}_{z_i\tau}$ is time-dependent.

Below we generalize the probabilistic model in Equation (4.2) to incorporate the traditional time-independent duration model. In stochastic RCPSP, it is assumed that the duration of a_i can be represented as a random variable $\boldsymbol{Y}_i$ that is not conditioned on s_i. Hence, the duration uncertainty model is a random vector $\boldsymbol{Y} = (\boldsymbol{Y}_1, ..., \boldsymbol{Y}_N)$. In reality, it is possible that both $\boldsymbol{X}$ and $\boldsymbol{Y}$ exist at the same time, since they reflect different sources of uncertainty. In this case, the baseline duration of a_i is not fixed as d_i^s, instead it is the time-independent random variable $\boldsymbol{Y}_i$. Then $\boldsymbol{D}_i$ conditions on s_i and $\boldsymbol{Y}_i$ at the same time, and its CDF can be written as:

$$\begin{aligned} F_{\boldsymbol{D}_i}(d_i \mid s_i = t, \boldsymbol{Y}_i = y_i) &= P(\boldsymbol{D}_i \leq d_i \mid s_i = t, \boldsymbol{Y}_i = y_i) \\ &= P\left(\sum_{\tau=t}^{t+d_i-1} \boldsymbol{X}_{z_i\tau} \geq y_i \middle| s_i = t, \boldsymbol{Y}_i = y_i \right), \end{aligned} \tag{4.3}$$

[2]The binary assumption is somehow restrictive; however our approach can be easily adapted to support more "fine-grained" models where the domain of x_{zt} has more values.

where y_i is a possible realization of $\boldsymbol{Y}_i$. We denote the uncertainty model studied here as $\boldsymbol{U} =< \boldsymbol{X}, \boldsymbol{Y} >$, which is a tuple with two components. Clearly, when the baseline durations are deterministic, i.e. $P(\boldsymbol{Y}_i = d_i^s) = 1$ holds for all activities, Equation (4.3) is equivalent to Equation (4.2) when $\boldsymbol{Y}_i = d_i^s$ since the dependence on $\boldsymbol{Y}_i$ can be removed. Meanwhile, we can have the following observation:

Observation 4.1 *When $P(\boldsymbol{X}_{zt} = 1) = 1$ holds for all z and t, the random duration $\boldsymbol{D}_i$ specified by Equation (4.3) is time-independent, and shares the same distribution with $\boldsymbol{Y}_i$.*

The correctness of Observation 4.1 can be verified as follows:

$$\begin{aligned} F_{\boldsymbol{D}_i}(d_i \mid s_i = t, \boldsymbol{Y}_i = y_i) &= P(\boldsymbol{D}_i \le d_i \mid s_i = t, \boldsymbol{Y}_i = y_i) \\ &= P\left(\sum_{\tau=t}^{t+d_i-1} \boldsymbol{X}_{z_i\tau} \ge y_i \middle| s_i = t, \boldsymbol{Y}_i = y_i \right) \\ &= P(d_i \ge y_i | s_i = t, \boldsymbol{Y}_i = y_i) = P(\boldsymbol{Y}_i \le d_i). \end{aligned} \tag{4.4}$$

Therefore, the uncertainty model $\boldsymbol{U} =< \boldsymbol{X}, \boldsymbol{Y} >$ and the CDF in Equation (4.3) can be used to describe the random activity duration with both the traditional time-independent uncertainty and the time-dependent workability uncertainty.

4.2.2 The Proactive Problem

Now, we are ready to formulate the proactive problem studied in this chapter: given a RCPSP instance and the uncertainty model $\boldsymbol{U}$, find a POS $G_R^* \in \boldsymbol{G}_R$ that minimizes the expected makespan:

$$G_R^* = \underset{G_R \in \boldsymbol{G}_R}{\operatorname{argmin}} \{ g(G_R) = \mathbb{E}[MS(G_R, \boldsymbol{U})] \}, \tag{4.5}$$

where $\mathbb{E}[\cdot]$ is the expectation operator and $MS(G_R, \boldsymbol{U})$ is a random variable representing the (stochastic) makespan of a solution G_R on $\boldsymbol{U}$. Note that when the uncertainty model $\boldsymbol{U}$ only contains component $\boldsymbol{Y}$, the problem in Equation (4.5) reduces to the traditional stochastic RCPSP.

Equation (4.5) is a hard stochastic optimization problem, not only due to the combinatorial nature of RCPSP. In fact, even evaluating a solution is intractable. When $\boldsymbol{U}$ only contains component $\boldsymbol{Y}$, the expected value computation of a given solution G_R is equivalent to the MEAN PERT problem (Hagstrom, 1988), which is

shown to be #P-complete. When $\boldsymbol{U}$ only contains component $\boldsymbol{X}$, the number of possible realizations of $\boldsymbol{X}$ is 2^{ZT}, which grows exponentially with the problem size. To circumvent the hardness in computing the expected makespan, we use SAA to approximate the problem in Equation (4.5) in the next Section.

4.3 Sample Average Approximation

Sample Average Approximation (SAA) is a Monte-Carlo simulation based approach for approximately solving hard discrete stochastic optimization problems (Kleywegt et al., 2002). The basic idea of SAA is very intuitive. Essentially, a set of independent random samples are drawn from the distribution, and then a SAA problem, which is an approximation of the original problem, is formulated by substituting the original objective function (i.e. expected value) with the sample average function. By solving the SAA problem, an approximate solution can be found, which is proved to converge to the optimal solution of the original problem at an exponential rate with the increase of sample size (number of samples).

Now we show how to approximate the problem in Equation (4.5) using SAA. Given the uncertainty model $\boldsymbol{U} =< \boldsymbol{X}, \boldsymbol{Y} >$, a sample is a tuple $u =< X, Y >$, where $X = [x_{zt}]_{Z \times T}$ and $Y = (y_1, ..., y_N)$ is a realization of $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively. For convenience, we use $x(u, z, t)$ to represent the workability of activity type z in time slot t specified by sample u, and let $y(u, i)$ to be the baseline duration of activity a_i in u. We first draw Q random samples $\boldsymbol{u} = \{u^1, ..., u^Q\}$ independently from $\boldsymbol{U}$. Then, the SAA problem of Equation (4.5) can be formulated as:

$$\hat{G}_R^* = \underset{G_R \in \boldsymbol{G}_R}{\operatorname{argmin}} \left\{ \hat{g}(G_R) = \frac{1}{Q} \sum_{q=1}^{Q} MS(G_R, u^q) \right\}, \tag{4.6}$$

where $\hat{G}_R^*$ is the optimal solution of the SAA problem, $\hat{g}(G_R)$ is the sample average function of the original expected value function $g(G_R)$ in Equation (4.5), and $MS(G_R, u^q)$ is the makespan of a solution G_R on sample u^q. As proved in (Kleywegt et al., 2002), $\hat{G}_R^*$ will converge to G_R^* at an exponential rate with the increase of Q. Moreover, Equation (4.6) is a deterministic problem instead of a stochastic one, which can help to avoid reasoning on the complex activity duration models and facilitate the design of solution algorithms.

Next, we show how to compute the sample average function $\hat{g}(G_R)$ for a given solution G_R. More specifically, we only need to show the computation of $MS(G_R, u)$. According to Equation (4.1), the start time s_i of an activity a_i can be determined by the completion times of all its predecessors specified by a solution G_R. Therefore, we only need to determine the durations of these predecessors on sample u. Intuitively, the duration of an activity on a sample should be also time-dependent. Specifically, for activity a_i, if it starts at s_i on a sample u, then a duration d_i is feasible if a_i can obtain enough workable time slots before completion, i.e. $\sum_{\tau=s_i}^{c_i-1} x(u, z_i, \tau) \geq y(u, i)$, where $c_i = s_i + d_i$ is the completion time. Many d_i values can satisfy the above condition, but we can show that it is sufficient to use the minimum value of them, as given by the following equation:

$$d_i(s_i, u) = \min \left\{ d > 0 \middle| \sum_{\tau=s_i}^{s_i+d-1} x(u, z_i, \tau) \geq y(u, i) \right\}. \tag{4.7}$$

In other words, an activity should be completed once it acquires enough workable time slots. By definition, $d_i(s_i, u)$ is the smallest feasible duration. To show the rationale, we first prove the following lemma:

Lemma 4.1 *For an activity a_i and a sample u, given two start times s_i^1 and s_i^2, if $s_i^1 \leq s_i^2$, then for any feasible duration d_i^2 of s_i^2, $c_i^1(s_i^1, u) \leq c_i^2$ holds, where $c_i^1(s_i^1, u) = s_i^1 + d_i(s_i^1, u)$ and $c_i^2 = s_i^2 + d_i^2$.*

Proof. We only need to show that $c_i^1(s_i^1, u) \leq c_i^2(s_i^2, u) = s_i^2 + d_i(s_i^2, u)$, since $d_i(s_i^2, u)$ is less than any other feasible d_i^2. For convenience, below we denote $c_i^1(s_i^1, u)$ and $c_i^2(s_i^2, u)$ as $c_i(1)$ and $c_i(2)$, respectively. According to Equation (4.7),

$$\sum_{t=s_i^1}^{c_i(1)-1} x(u, z_i, t) = \sum_{t=s_i^2}^{c_i(2)-1} x(u, z_i, t) = y(u, i). \tag{4.8}$$

It is easy to verify that the lemma holds if $s_i^2 \geq c_i(1)$. When $s_i^1 \leq s_i^2 < c_i(2)$, we first assume $c_i(1) > c_i(2)$. Then, we have

$$\sum_{t=s_i^1}^{s_i^2-1} x(u, z_i, t) + \sum_{t=s_i^2}^{c_i(2)-1} x(u, z_i, t) + \sum_{t=c_i^2}^{c_i(1)-1} x(u, z_i, t) = y(u, i). \tag{4.9}$$

Since the second term in the left hand side of Equation (4.9) equals to $y(u, i)$, we have $\sum_{t=s_i^1}^{s_i^2-1} x(u, z_i, t) + \sum_{t=c_i(2)}^{c_i(1)-1} x(u, z_i, t) = 0$, which indicates that $\sum_{t=c_i(2)}^{c_i(1)-1} x(u, z_i, t) = 0$

since $x(u, z_i, t) \geq 0$. Hence, the third term in the left hand side of Equation (4.9) can be removed, indicating $d'_i = c_i(2) - s_i^1$ is a feasible duration. However, based on the assumption, $d'_i < c_i(1) - s_i^1 = d_i(s_i^1, u)$, which contradicts Equation (4.7) which states that $d_i(s_i^1, u)$ is the minimum feasible duration. □

Based on Lemma 4.1, we can prove the following proposition:

Proposition 4.1 *Given a POS G_R and a sample u, the schedule $S(G_R, u)$ generated by using Equations (4.1) and (4.7) produces the lowest makespan.*

Proof. Let $S'(G_R, u)$ be a schedule obtained by setting the duration of an activity a_i to a feasible duration $d'_i > d_i(s_i, u)$. Then according to Equation (4.1), the start time of any immediate successor a_j of a_i cannot be earlier than the corresponding start time in $S(G_R, u)$. According to Lemma 4.1, the finish time of j determined by $S'(G_R, u)$ cannot be earlier than that determined by $S(G_R, u)$, which indicates a non-negative delay of a_j. By further propagating this delay through G_R, a makespan equal or larger than $MS(S(G_R, u))$ will be obtained. □

It should be noted that Equation (4.7) does not exclude the possibility that an activity could obtain a smaller duration by starting later, due to the time-dependent workability uncertainty. However, Lemma 4.1 and Proposition 4.1 show that for activities in a POS, it is not helpful to start late, since the delay will be propagated to the "downstream" activities and finally lead to a non-negative increase of the makespan. Further, we can have the following observation which will be used in designing our algorithms:

Observation 4.2 *Given two POS G_R^1 and G_R^2, if $V(G_R^1) = V(G_R^2) = A_p$ and $E(G_R^1) \subseteq E(G_R^2)$, then $MS(G_R^1, u) \leq MS(G_R^2, u)$ holds for any sample u.*

The reason is that, for any activity $a_i \in A_p$, we can see that $Pre(a_i, G_R^1) \subseteq Pre(a_i, G_R^2)$. Therefore, the start time of a_i in schedule $S(G_R^2, u)$ cannot be earlier than that in $S(G_R^1, u)$, according to Equation (4.1). This indicates that a_i cannot complete earlier by using G_R^2 than using G_R^1, according to Lemma 4.1. Thus, G_R^2 results in an equal or larger makespan than G_R^1 on u.

Let $MS(G_R, u) = MS(S(G_R, u))$ be the makespan of G_R on u. The value of $MS(G_R, u)$ can be computed in many efficient ways. In Algorithm 3, we give a simple algorithm with a complexity of $\mathcal{O}(N^2T)$. Therefore, it is tractable to evaluate

Algorithm 3: ComputeMakespan(G_R, u)

Input: G_R: a solution; u: a sample

Output: $MS(G_R, u)$: the makespan of G_R on u

```
CS ← {a_0};
while |CS| < |A_p| do
    ES ← {a_i ∉ CS | Pre(a_i) ⊆ CS} ;
    foreach a_i ∈ ES do
        s_i ← max{c_j | a_j ∈ Pre(a_i)} ;
        c_i ← s_i + d_i(s_i, u) ;
        CS ← CS ∪ {a_i} ;
return MS(G_R, u) ;
```

the objective $\hat{g}(G_R)$, with a complexity of $\mathcal{O}(MN^2T)$. However, the SAA problem is intractable, as stated below:

Proposition 4.2 *The SAA problem in Equation (4.6) is NP-hard.*

Proof. We follow the proof for deterministic RCPSP (Blazewicz et al., 1983), where it is reduced from a NP-complete problem Partition Into Triangles (PIT): for a graph $G = (V, E)$ where $|V| = 3t$, is there a partition of G into t disjoint subsets, such that each subset contains three pairwise adjacent vertices?

For any PIT instance, we first construct a RCPSP instance as in (Blazewicz et al., 1983). Firstly, for each $i \in V$ we create an activity a_i. Next, for each pair $(i, j) \notin E$, a resource r_{ij} with capacity $C_{ij} = 1$ is added, which is only required by a_i and a_j with $b_i^{ij} = b_j^{ij} = 1$, and $b_l^{ij} = 0$ for other activities a_l. Then we construct an instance for the SAA problem, by adding one sample u where $x_{zt} = 1$ for all z and t, and $y_i = 1$ for all a_i. We claim that the SAA problem has a solution G_R with $\hat{g}(G_R) \leq t$ if and only if the PIT instance has a solution.

If we can find a solution to the PIT instance, then we immediately have a schedule S with $MS(S) \leq t$. From S, a feasible G_R can be constructed by sequencing the activities on each resource and adding a resource carrying edge from one activity to its immediate successor in the sequence. Propagating this G_R on u will produce a schedule with the makespan $MS(G_R, u) \leq t$, hence $\hat{g}(G_R) \leq t$. On the other hand, if we can find a G_R satisfying $\hat{g}(G_R) \leq t$, then the schedule $S(G_R, u)$ must satisfy

$MS(G_R, u) \leq t$, indicating that the PIT instance has a feasible solution. □

Since the SAA problem is intractable, we design two branch-and-bound algorithms in the next two sections to solve it efficiently.

4.4 The Flow-based Algorithm

In this section, we design a branch-and-bound algorithm to solve the SAA problem, which searches for the optimal POS by constructing feasible resource flows. We first analyze the relations between POS and AON-flow Network.

4.4.1 Relations between POS and AON-flow Network

As we have mentioned in Section 4.1, an AON-flow Network is also a POS, since all the possible resource conflicts are resolved by the resource flows. However, the reverse relation, i.e. whether an AON-flow Network can be obtained from a POS, is not straightforward. Below we analyze this relation in detail.

We first introduce a method for finding a feasible flow in a given DAG $G = (A_p, E)$. When there is only one resource r with capacity C, it has been shown in (Leus and Herroelen, 2004) that the existence of a feasible flow for G can be checked by computing a maximum flow in a transformed network G^T constructed as follows: 1) create two vertices a_i^s and a_i^t for each $a_i \in A$, and one vertex for a_0 and a_{N+1} named as a_0^s and a_{N+1}^t, respectively; 2) create two vertices, s and t with an edge (t, s) as the virtual source and sink, and add edges (s, a_i^s), (a_i^t, t) for all $a_i \in A_p$; 3) for each $(a_i, a_j) \in E(G)$, add an edge (a_i^s, a_j^t). Each (s, a_i^s) and (a_i^t, t) has a capacity b_i that is equal to the resource requirement of a_i, while the capacities of other edges are $+\infty$. An example of this transformation is shown in Figure 4.7. Let $f(G^T)$ be the maximum (s, t) flow value in G^T, then there exists an AON-flow Network G_F with $E(G_F) \subseteq E(G)$ if and only if $f(G^T) = f_{max}$, where $f_{max} = C + \sum_{a_i \in A} b_i$. Moreover, a feasible flow in G can be obtained by setting f_{ij} to the flow value on the edge (a_i^s, a_j^t) in G^T. Furthermore, based on the well-known integral flow theorem, there exists an optimal integer flow, i.e. all f_{ij} are integers. This integer maximum flow can be found very efficiently using maximum flow algorithms (e.g. Edmonds-Karp algorithm).

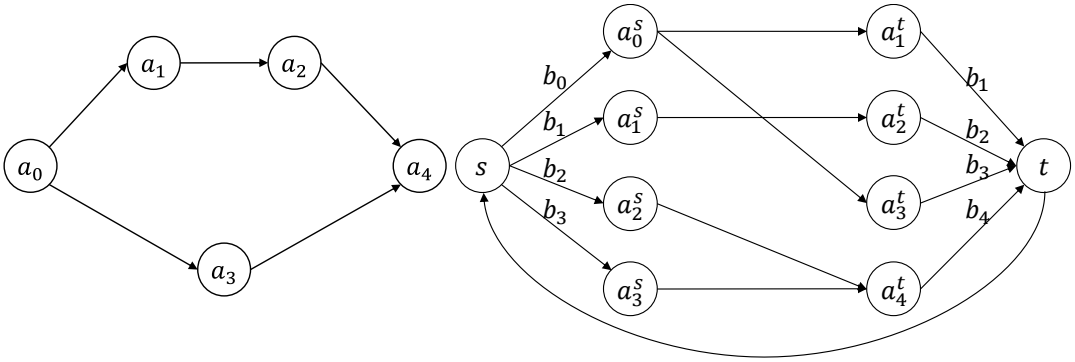


Figure 4.7: An Example of Network Transformation (left: original DAG G_V; right: transformed network G'_V, where integers beside edges represent capacities)

Here we extend the above procedure to support multiple resources. For each $r_k \in R$, we maintain a transformed network $G^T(k)$ for a given DAG G. Note that these networks have the same edge sets, while the edge capacities are set to b_i^k for the corresponding $G^T(k)$. Let $f_{max}^k = C_k + \sum_{a_i \in A} b_i^k$ for r_k, then we can conclude that there exists an AON-flow Network G_F with $E(G_F) \subseteq E(G)$ if and only if $f(G^T(k)) = f_{max}^k$ holds for all $r_k \in R$. Furthermore, we can show that whether a DAG is a POS can be checked in polynomial time, by checking the existence of AON-flow Network.

Proposition 4.3 *For any POS $G_R \in \boldsymbol{G}_R$, there must be an AON-flow Network $G_F \in \boldsymbol{G}_F$ such that $E(G_F) \subseteq E(G_R)$.*

Proof. If no such AON-flow Network exists, then there must be a resource $r_k \in R$ with $f(G_R^T(k)) < f_{max}^k$. This means there must be some activity a_i which cannot secure enough amount of r_k by the edges in $E(G_R)$, since the flow in $G_R^T(k)$ is already maximized. Hence in the actual execution, it is possible that r_k is not enough for a_i to start at the time determined by G_R, which implies that potential precedence constraints are needed to resolve resource conflicts. □

Proposition 4.3 enables us to search for the optimal POS by searching in the space of AON-flow Networks, which is to solve the following problem[3]:

$$\hat{G}_F^* = \underset{G_F \in \boldsymbol{G}_F}{\operatorname{argmin}} \left\{ \hat{g}(G_F) = \frac{1}{Q} \sum_{q=1}^{Q} MS(G_F, u^q) \right\}. \tag{4.10}$$

[3]Note that the computation of $g(G_F)$ and $MS(G_F, u^q)$ follow the same procedure as that for POS, since only the precedence relations in $E(G_F)$ are needed.

This is can be justified by the following conclusion:

Proposition 4.4 *For any optimal solution $\hat{G}_F^*$ of the problem in Equation (4.10), the corresponding POS $\hat{G}_R^* = (A_p, E(\hat{G}_F^*))$ solves the problem in Equation (4.6) optimally.*

Proof. Clearly we have $\hat{g}(\hat{G}_F^*) = \hat{g}(\hat{G}_R^*)$. Suppose there is another POS $G'_R \neq \hat{G}_R^*$ that has a lower SAA objective value than $\hat{G}_R^*$, i.e. $\hat{g}(G'_R) < \hat{g}(\hat{G}_R^*)$. Then according to Proposition 4.3, there must exist an AON-flow Network G'_F with $E(G'_F) \subseteq E(G'_R)$. Based on Observation 4.2, for any sample u, we have $MS(G'_F, u) \leq MS(G'_R, u)$, leading to $\hat{g}(G'_F) \leq \hat{g}(G'_R) < \hat{g}(\hat{G}_R^*) = \hat{g}(\hat{G}_F^*)$. This indicates that in the space of AON-flow Networks $\boldsymbol{G}_F$, G'_F is a better solution than $\hat{G}_F^*$, which contradicts the fact that $\hat{G}_F^*$ is optimal. □

Next, we introduce our algorithm for solving the problem in Equation (4.10).

4.4.2 Branching Scheme

Our flow-based algorithm, named `BnB-Flow`, is a depth-first tree search process that directly exploits the feasible domain $\boldsymbol{G}_F$ to find the optimal solution G_F^*. Each search node is associated with a partial solution G'_F that contains a subset of activities, i.e. $V(G'_F) \subseteq A_p$, and each of them is provided enough resources by the incoming resource-carrying edges in G'_F. The branch-and-bound process is based on a two-level branching scheme to determine the next activity to be linked to the partial solution, along with the corresponding edges. The first level is called the *activity level*, where an unlinked and precedence feasible activity will be selected for branching. The second level is called the *link level*, where a resource and precedence feasible link of the chosen activity will be selected for branching.

The branch-and-bound process can be described by the pseudo code in Algorithm 4, which shows a BnB_Flow function that will be called on each search node. This function starts with identifying a set ES of activities that are eligible for being linked to the partial solution G'_F. An activity is feasible if it is not included in G'_F, but all its immediate predecessors are, therefore $ES = \{a_i \in A_p | a_i \notin V(G'_F), Pre(a_i) \in V(G'_F)\}$. If ES is empty, then all activities are linked to G'_F and a feasible solution is reached. Then the algorithm computes the SAA objective of the solution G'_F, updates the best solution, and backtracks (Lines 3-7). If ES is not empty, then the

algorithm enters the two-level branching process. In the activity level, an eligible activity $a_l \in ES$ will be chosen and removed from ES (Lines 9-10) for branching, until ES is empty. The activity can be chosen based on any criterion without affecting the correctness of the algorithm, but certain heuristic for activity selecting may help in reducing the computational time. We will further discuss the branching heuristics in Section 4.4.5. Once a_l is chosen, the algorithm computes the lower bound of this branching choice (link a_l to G'_F) (Line 11). If the lower bound is greater than or equal to the current best objective value $\hat{g}^*$, the search path is pruned; otherwise the algorithm enters the link level.

The first step in the link level is to identify all the feasible links for incorporating the chosen activity a_l to G'_F, and put them into set LK (Line 13) as branching candidates. A link $lk = \{(a_i, a_l)|a_i \in V(lk)\}$ is a set of edges that link a set of vertices $V(lk) \subseteq V(G'_F)$ in the partial solution G'_F to the chosen activity a_l. The approach of generating LK will be further discussed in Section 4.4.3. Then, similar to the activity level, a feasible link lk will be chosen and removed from LK for branching, until LK is empty (Lines 15-20). Once a link lk is chosen, it will be used to link a_l to G'_F, by calling the function LinkActivity (Line 17). Then, a new partial solution $\bar{G}'_F = (V(G'_F) \cup \{a_l\}, E(G'_F) \cup lk)$ is obtained, and the algorithm continues by calling BnB on $\bar{G}'_F$. Upon backtracking in the link level, the function RemoveActivity (Line 19) will be called to conduct inverse operation as LinkActivity, in order to remove a_l and lk from $\bar{G}'_F$. The branching heuristic and lower bounds for the link level are embedded in the ChooseLink function shown in Algorithm 5, and will be discussed in Section 4.4.3.

In `BnB-Flow`, an outgoing capacity matrix $OC = [oc_{ik}]_{(N+2)\times K}$ is also maintained to record the remaining available resource amounts for each activity. Specifically, oc_{ik} is the current amount of resource r_k that can be transfered from a_i to another activity. Before executing the algorithm, OC is set to be the initial value OC^0, where for each $r_k \in R$, oc^0_{ik} is set to b^k_i for all $a_i \in A$, while oc^0_{0k} and $oc^0_{N+1,k}$ are set to C_k and 0, respectively. In the LinkActivity function of Algorithm 4, if a_l is linked to G'_F using a link lk, then for each $r_k \in R$ and $a_i \in V(lk)$, oc_{ik} will be set to $oc_{ik} - f_{ilk}$ if the edge (a_i, a_l) carries positive flow for r_k. Accordingly, the reverse operation will be conducted by the RemoveActivity function upon backtracking. `BnB-Flow` is invoked by calling BnB_Flow$(G'^0_F, null, L, OC^0)$, where $G'^0_F = (\{0\}, \emptyset)$ is the initial

Algorithm 4: BnB_Flow($G'_F, \hat{G}^*_F, \hat{g}^*, OC$)

Input: G'_F: current partial solution; $\hat{G}^*_F$: current best solution; $\hat{g}^*$: current best objective value; OC: outgoing capacity matrix

```
1  ES ← FindEligibleActivities(V(G'_F)) ;
2  if ES = ∅ then
3      ĝ' ←ComputeObj(G'_F) ;
4      if ĝ' < ĝ* then
5          ĝ* ← ĝ';
6          Ĝ*_F ← G'_F;
7      return;
8  while ES ≠ ∅ do
9      a_l ← ChooseActivity(ES);
10     ES ← ES \ {a_l};
11     LB(G'_F, a_l) ← ComputeLB_A(G'_F, a_l);
12     if LB(G'_F, a_l) < ĝ* then
13         LK ← FindFeasibleLinks(G'_F, a_l, OC);
14         lk ←ChooseLink(G'_F, LK, ĝ*);
15         while lk ≠ null do
16             LK ← LK \ {lk};
17             Ḡ'_F ←LinkActivity(a_l, G'_F, lk, OC);
18             BnB_Flow(Ḡ'_F, Ĝ*_F, ĝ*, OC);
19             G'_F ←RemoveActivity(a_l, Ḡ'_F, lk, OC);
20             lk ←ChooseLink(G'_F, LK, ĝ*);
21 return;
```

partial solution which contains only the dummy start activity a_0 and L is a large double value. `BnB-Flow` is complete when the lower bounds are admissible (i.e. they never overestimate the best objective that can be achieved by the subtree rooted from the corresponding search node), since the solution domain $\boldsymbol{G}_F$ is completely exploited.

4.4.3 Finding and Choosing Feasible Links

In this section, we describe the approach of identifying and selecting the feasible links for a chosen activity. We begin by defining the feasibility of a link. Given a partial solution G'_F and an unlinked activity a_l, a link lk is said to be feasible if the following conditions are satisfied by the new partial solution $\bar{G}'_F = (V(G'_F) \cup \{a_l\}, E(G'_F) \cup lk)$:

$$(a_i, a_l) \in Tr(\bar{G}'_F), \forall a_i \in Pre(a_l) \tag{4.11}$$

$$\sum_{(a_i,a_l)\in lk} f_{ilk} = b_l^k, \forall r_k \in R. \tag{4.12}$$

The first condition in Equation (4.11) guarantees the original precedence constraints in G_p is respected by $\bar{G}'_F$. The second condition in Equation (4.12) requires that a_l must obtain enough resources from all the edges in lk. Below we give an observation that enables us to limit the search space to integer resource flows only.

Observation 4.3 *For all k and i, if $C_k \in \mathbb{N}$ and $b_i^k \in \mathbb{N}$, then it is sufficient to consider only integer flows.*

The correctness of this observation can be justified as follows. For any AON-flow Network G_F, clearly the DAG $G_R = (A_p, E(G_F))$ is a POS. According to Proposition 4.3, there must be an AON-flow Network G'_F with $E(G'_F) \subseteq E(G_R)$. As we mentioned in Section 4.4.1, the flow values in G'_F should be integers. Based on the same procedure in the proof of Proposition 4.4, we have $\hat{g}(G'_F) \leq \hat{g}(G_F)$. This means for any G_F, there must exist a G'_F with integer flows that has a better SAA objective value.

Here we use an enumeration approach to generate the set LK. Firstly, all activities $a_i \in G'_F$ with positive oc_{ik} values are identified as the candidates for linking a_l. Then, all the links that satisfy the condition in Equation (4.12) and contain only positive resource flows are enumerated to form the set LK. Finally, each element $lk \in LK$ is checked against the condition in Equation (4.11). If any immediate predecessor $a_i \in Pre(a_l)$ cannot reach a_l by lk, i.e. $(a_i, a_l) \notin Tr(\bar{G}'_F)$, an additional edge (a_i, a_l) with zero resource flow is incorporated in lk to make it precedence feasible. Note that this does not violate the positive flow condition of AON-flow Network, since (a_i, a_l) simply represents a precedence constraint in E_p.

Algorithm 5: ChooseLink($G'_F, LK, \hat{g}^*$)

Input: G'_F: current partial solution; LK: current set of feasible links; $\hat{g}^*$: current best objective value

Output: lk: the chosen edge

```
1 while LK ≠ ∅ do
2     lk ←GetLink(LK);
3     LB(G'_F, lk) ←ComputeLB_L(G'_F, lk);
4     if LB(G'_F, lk) < ĝ* then
5         return lk;
6     else
7         LK ←RemoveLinks(LK, lk);
8 return null;
```

The branching and pruning process in the link level is shown in Algorithm 5. When LK is not empty, an element is selected based on certain criterion (will be further discussed in Section 4.4.5) in Line 2. Then in Lines 3-5, the lower bound of this branching alternative (i.e. link a_l to $\bar{G}'_F$ using lk) is computed and compared with the current best objective value $\hat{g}^*$ to determine if the search path should be pruned or not. If not, lk will be returned to Algorithm 4 for branching.

Due to the combinatorial nature, the enumeration operations may produce many branching alternatives. Here we design an additional pruning step in Lines 6-7 of Algorithm 5 to further reduce the size of LK. Essentially, whenever a link lk is pruned, then any link $lk' \in LK$ satisfying $V(lk) \subseteq V(lk')$ can also be safely pruned, since they can only result in equal or larger lower bound values than that of lk. The rationale is based on the following observation:

Observation 4.4 *Given two links lk^1 and lk^2 for linking a_l to G'_F, if $V(lk^1) \subseteq V(lk^2)$, then $LB(G'_F, lk^1) \leq LB(G'_F, lk^2)$.*

Observation 4.4 will be justified in Section 4.4.4 when the lower bounding function ComputeLB_L is discussed. Note that not all admissible lower bounds satisfy this observation, but the one we design in Section 4.4.4 does.

4.4.4 Lower Bounds

In this section, we introduce the lower bounds we designed for the two branching levels, i.e. ComputeLB_A for the activity level, and ComputeLB_L for the link level. To guarantee the optimality, these two lower bounds must be admissible. Before introducing the lower bounding technique, we first give a general lower bound on the sample average function defined in Equation (4.6). Given a partial solution G'_F and a branching alternative Δ (either an activity or a link), it is straightforward to verify that the LB function defined in the equation below is an admissible lower bound of $\hat{g}$:

$$LB(G'_F, \Delta) = \frac{1}{Q} \sum_{q=1}^{Q} MS_{LB}(G'_F, \Delta, u^q), \tag{4.13}$$

where $MS_{LB}(G'_F, \Delta, u^q)$ is a lower bound of the makespan of choosing Δ on sample u^q. In other words, to compute the lower bound of Δ on $\hat{g}$, we only need to compute its lower bound on each individual sample u^q. Next, we give an observation based on the properties of POS and SAA problem:

By leveraging Equation (4.13) and Observation 4.2, we construct the two lower bounds based on the critical path lower bound for solving deterministic RCPSP (Demeulemeester and Herroelen, 1997). Essentially, for the unlinked activities $a_i \notin V(G'_F)$, only the original precedence constraints in E_p are considered in computing the lower bounds. Below we first discuss the lower bounding computation at the link level.

ComputeLB_L. Given a partial solution G'_F, to compute the lower bound of a feasible link lk, we construct an auxiliary graph $\bar{G}''_F$ and compute its makespan on each sample u^ξ. Specifically, $\bar{G}''_F$ is an augmented graph of $\bar{G}'_F$ where the unlinked activities $a_i \notin V(\bar{G}'_F)$ are linked using the edges in E_p, i.e. $\bar{G}''_F = (A_p, E(\bar{G}'_F) \cup E_p)$. Then $\bar{G}''_F$ is propagated on each sample u^ξ to obtain a temporal feasible schedule $S(\bar{G}''_F, u^\xi)$, where $a_i \notin \bar{G}'_F$ is not necessarily resource feasible. We can conclude that the LB value defined in the below equation is an admissible lower bound of the branching choice of linking a_l to G'_F using lk:

$$LB(G'_F, lk) = \frac{1}{Q} \sum_{q=1}^{Q} MS(\bar{G}''_F, u^q). \tag{4.14}$$

This is because for any feasible solution $G_F \in \boldsymbol{G}_F$ obtained by extending $\bar{G}'_F$, we have $E(\bar{G}''_F) \subseteq E(G_F)$ since additional edges are added to resolve resource conflicts. According to Observation 4.2, $MS(\bar{G}''_F, u^q)$ is a lower bound of the makespan obtained by using G_F on u^q, indicating $LB(G'_F, lk) \leq \hat{g}(G_F)$ according to Equation (4.13).

Now we show the the correctness of Observation 4.4. Given two links lk^1 and lk^2 for linking the same activity a_l to a partial solution G'_F, we can construct two auxiliary graphs $\bar{G}''^1_F$ and $\bar{G}''^2_F$. If $V(lk^1) \subseteq V(lk^2)$, then we have $E(\bar{G}''^1_F) \subseteq E(\bar{G}''^2_F)$. According to Observation 4.2, $LB(G'_F, lk^1) \leq LB(G'_F, lk^2)$.

ComputeLB_A. Different from ComputeLB_L, we cannot construct a common solution and propagate it on all samples to compute a lower bound, since the feasible links have not been identified yet. Below we take a different approach to compute the lower bound of linking an activity a_l to a partial solution G'_F. We first construct an auxiliary graph $G'^r_F = (A^r_p, E^r)$ for the unlinked activities except a_l, where $A^r_p = A_p \setminus (V(G'_F) \cup \{a_l\})$ and $E^r = \{(a_i, a_j) \in E | a_i, a_j \in A^r_p\}$. Then, for each sample u, we propagate G'_F on it to obtain a schedule $S(G'_F, u)$ that contains only activities in $V(G'_F)$. Based on $S(G'_F, u)$, we compute the earliest precedence and resource feasible start time s_l of a_l, along with the duration $d_l(s_l, u)$ as defined in Equation (4.7). Finally, we propagate the auxiliary graph G'^r_F on u to obtain a complete schedule that contains all activities in A_p. The propagation of G'^r_F should consider $S(G'_F, u)$, $c_l = s_l + d_l(s_l, u)$ and a set of precedence constraints $E^l = \{(a_i, a_j) \in E | a_i \notin A^r_p, a_j \in A^r_p\}$. Let $S(G'_F, a_l, u)$ be the schedule obtained in this way on sample u. Then we can conclude that $MS(G'_F, a_l, u) = MS(S(G'_F, a_l, u))$ is a lower bound for the makespan of any feasible solution $G_F \in \boldsymbol{G}_F$ obtained by extending any $\bar{G}'_F$, where $\bar{G}'_F$ is a partial solution obtained by incorporating a_l to G'_F using a feasible link. This is because for any G'_F and u, we have $\forall S(\bar{G}'_F, u)$ since a_l cannot start earlier than s_l on u. According to Lemma 4.1, $MS(G'_F, a_l, u) \leq MS(\bar{G}'_F, u) \leq MS(G_F, u)$. Therefore, the LB value defined in the below equation is an admissible lower bound of incorporating a_l to G'_F using any feasible link:

$$LB(G'_F, a_l) = \frac{1}{Q} \sum_{q=1}^{Q} MS(G'_F, a_l, u^q). \tag{4.15}$$

4.4.5 Branching Heuristics

In this section, we introduce the heuristics for selecting the branching alternatives in `BnB-Flow`. In general, we aim at finding high-quality solutions as early as possible, so that more search space can be pruned.

For the activity choosing step in Line 9 of Algorithm 4, we adopt two priority rules, Maximum Total Successors (MTS) and minimum Latest Finish Time (LFT), which are commonly used for solving deterministic RCPSP. These two rules are experimentally shown to be able to produce good solutions with heuristic schedule generation schemes Kolisch (1996), in which the activities are scheduled in an order determined by these priority rules. When used for choosing activity from the eligible set ES, MTS gives priority to the one with more number of immediate successors, while LFT prefers the activity with smaller LFT value. For a given activity, the number of total immediate successors can be easily determined by the AON network G_p, and the LFT value can be computed by critical path method Kolisch (1996).

For the link choosing step in Line 5 of Algorithm 5, we design two heuristics, minimum *Average Earliest Start Time* (AEST) and *Minimum Link Predecessors* (MLP) based on the properties of the SAA problem. Specifically, AEST is designed according to Lemma 4.1, which prefers the link lk with smaller average earliest start time $aest(lk)$ on all samples. To compute $aest(lk)$, we first compute the earliest start time $est(lk, u^q)$ of the chosen activity on each sample u^q, then take the average value, i.e. $aest(lk) = 1/Q \cdot \sum_{q=1}^{Q} est(lk, u^q)$. The intuition of designing MLP is based on Observation 4.4, which prefers a link with a smaller number of vertices, i.e. $|V(lk)|$.

4.5 The MCS-based Algorithm

Different from `BnB-Flow`, our second algorithm, named `BnB-MCS`, directly searches for the optimal solution in the space of POS. This is done by iteratively detecting and resolving possible resource conflicts represented as *Minimal Critical Set* (MCS), which will be introduced in the section below.

4.5.1 Detecting and Resolving Minimal Critical Sets

We begin with the definitions of Critical Set and Minimal Critical Set, following the definitions in (Lombardi and Milano, 2012).

Definition 4.1 *Given an instance of RCPSP, for an augmented DAG G_V of the AON network G_p, a set of activities $A_c \subseteq A$ is a Critical Set (CS) of resource r_k, if (a) $\sum_{a_i \in A_c} b_i^k > C_k$ and (b) $\forall a_i, a_j \in A_c$, $(a_i, a_j) \notin Tr(G_V)$ and $(a_j, a_i) \notin Tr(G_V)$.*

In other words, activities in a CS may temporally overlap, and have a total resource requirement higher than the capacity. When no CS exists in a temporal network G_R, it is a POS where all the possible resource conflicts are resolved by the temporal constraints in $E(G_R)$. The definition of MCS is given below:

Definition 4.2 *A critical set A_{mc} is a Minimal Critical Set (MCS), if $\forall a_i \in A_{mc}$, $\sum_{a_j \in A_{mc} \setminus \{a_i\}} b_j^k \leq C_k$.*

Intuitively, a MCS is a CS satisfying the minimality condition, i.e. it is no longer a CS if any activity is removed from it. Therefore, an MCS A_{mc} can be *resolved* by adding a precedence relation between any pair of activities (a_i, a_j) in it, which is called a *resolver* of A_{mc}. Let $Res(A_{mc}) = \{(a_i, a_j) | a_i, a_j \in A_{mc}, i \neq j\}$ be the set of all the possible resolvers of a MCS A_{mc}. Note that the resource conflict in CS may not be able to be resolved by adding precedence constraints for one pair of activities, since the minimality condition is not satisfied. Therefore, a CS should be reduced to a MCS to resolve the resource conflicts.

It has been shown in (Lombardi and Milano, 2012) that for a temporal network G and a resource r_k, the problem of detecting a possible CS is equivalent to the problem of routing the minimum amount of flow of r_k from source (a_0) to sink (a_{N+1}), such that the resource requirements b_i^k of all the activities $a_i \in A$ are satisfied. Further, this problem can be solved by solving a minimum flow problem on a transformed network $G^M(k)$, which can be solved in polynomial time using the inverse Ford-Fulkerson's algorithm (Lombardi and Milano, 2012). Denote this minimum flow as $f(G^M(k))$. When $f(G^M(k)) > C_k$, a CS A_c for r_k can be extracted by identifying all the activities in the source-sink cut, and $\sum_{a_i \in A_c} b_i^k = f(G^M(k))$. When $f(G^M(k)) \leq C_k$, all the possible conflicts for r_k has been resolved by $E(G)$. Therefore, starting from the AON network G_p, all resource conflicts can be resolved by iteratively

detecting and resolving MCS. This method is called Precedence Constraint Posting (PCP), and has already been applied in designing branch-and-bound approaches for deterministic RCPSP (Laborie, 2005; Lombardi and Milano, 2012; Lombardi et al., 2013)[4], where temporal reasoning can be applied for branching and constraint propagation. In contrast, our problem in Equation (4.5) is defined on multiple samples with time-dependent durations, which makes it very difficult to conduct the temporal reasoning. Therefore, we design the `BnB-MCS` algorithm, which purely reasons with resource constraints, except the lower bound computation.

4.5.2 Branching Scheme

Similar to `BnB-Flow`, `BnB-MCS` also employs a depth-first branch-and-bound searching process. Starting from the original AON network G_p, a POS is found by iteratively detecting and resolving MCS, until a conflict-free augmented DAG is obtained, which is a POS. For resource conflict detection, we adopt the method in (Lombardi and Milano, 2012) to detect CS, and then reduce it to MCS based on a heuristic procedure which will be further discussed in Section 4.5.4. In addition, we extend the constraint propagation procedure in (Leus and Herroelen, 2004), which is designed for single resource problems, to speed up the searching process. Since a POS must be acyclic, the set of feasible edges that can be added to G_p is $FS = \{(a_i, a_j) \notin E(G_p) | (a_j, a_i) \notin Tr(G_p)\}$. For each $(a_i, a_j) \in FS$, we maintain lower bound f^L_{ijk} and upper bound f^U_{ijk} of the (integer) flow f_{ijk} that can be imposed on it for r_k, with $0 \leq f^L_{ijk} \leq f^U_{ijk}$. Initially, $f^L_{ijk} = 0$ and $f^U_{ijk} = \min\{b^k_i, b^k_j\}$. During searching, these bounds will be tightened by constraint propagation. Let $sum^L_{ij} = \sum_{r_k \in R} f^L_{ijk}$ and $sum^U_{ij} = \sum_{r_k \in R} f^U_{ijk}$. Then $sum^L_{ij} > 0$ means there must be a flow on (a_i, a_j) while $sum^U_{ij} = 0$ indicates (a_i, a_j) cannot carry flow for any r_k. Based on the bound values and branching decisions, an edge $(a_i, a_j) \in FS$ has four status: 1) *included*, if $sum^L_{ij} > 0$; 2) *banned*, if $sum^U_{ij} = 0$; 3) *undecided*, if $sum^L_{ij} = 0$ and $sum^U_{ijk} > 0$; 4) *conflicted*, if $(a_j, a_i) \in Tr(G'_R)$ where G'_R is the current partial solution. We will further discuss how to maintain consistency of the flow bounds in Section 4.5.3.

Detail of this branching process is shown in the BnB_MCS function in Algorithm

[4]Though (Lombardi et al., 2013) aims at obtaining a POS for dynamic execution, it essentially solves a deterministic RCPSP where the duration of each activity is replaced by the expected value.

6. Inputs of the algorithm includes a partial solution $G'_R = (A_p, E_p \cup E'_R)$ which is an augmented DAG of G_p, the incumbent $\hat{G}'_R$ and its objective $\hat{g}^*$, and a set of edges BE that are banned by the current branching decisions. The first operation in Algorithm 6 is to detect an MCS A_{mc} in the input partial solution G'_R (Line 1). If A_{mc} is empty, then no resource conflict exists and a POS is found, hence the algorithm updates $\hat{G}^*_R$ and $\hat{g}^*$ if the found POS G'_R has a better objective value. Note that when a POS G_R is reached, the algorithm can backtrack safely. Because for any G'_R with $E(G_R) \subseteq E(G'_R)$, $MS(G_R, u) \leq MS(G'_R, u)$ holds for any sample u according to Observation 4.2, therefore $\hat{g}(G_R) \leq \hat{g}(G'_R)$ holds. If A_{mc} is not empty, then all its resolvers are retrieved as branching candidates. Specifically, these resolvers are ranked and put into a list $RES^L(A_{mc})$ according to some heuristic (Line 9).

In Lines 10-25, the ranked resolvers are selected for branching one by one. For a selected resolver, the algorithm first checks if it is *applicable* to G'_R, which requires it to be 1) not banned, 2) not implied by G'_R, and 3) not conflicted with G'_R (Lines 11-12). Only applicable resolvers will be considered for branching in Lines 13-25. For an applicable resolver, the algorithm can enforce two status to it, i.e., either included in or banned from G'_R. For the option of including, the algorithm first computes the lower bound of incorporating it into G'_R, and compare it with the incumbent value $\hat{g}^*$ to decide if the search path should be pruned or not (Line 13). If not, G'_R will be updated to include the resolver (Line 14). Further, all resources $r_k \in R$ will be ranked as a list R^L to conduct constraint propagation. The ranking heuristic will detailed in Section 4.5.4. More specifically, if r_k is chosen, we impose $f^L_{ijk} = 1$ and propagate it to maintain the bound consistency (Line 17). If the propagation is successful, the algorithm branches to the next level, otherwise the search path is pruned. If all resources have been tried, the chosen resolver will be removed from G'_R (Line 20) and the algorithm will try the option of banning the chosen resolver from G'_R, i.e. adding it to BE (Line 21). Then, we impose $f^U_{ijk} = 0$ (which automatically imposes $f^L_{ijk} = 0$) for all $r_k \in R$, and propagate it to maintain bound consistency. If the propagation is successful, the algorithm continues by calling BnB_MCS. Upon backtracking, the banned resolver will be removed from BE (Line 25). `BnB-MCS` is invoked by calling BnB_MCS$(G_p, null, L, \emptyset)$. Upon termination, the optimal POS can be found, if the lower bound is admissible.

Algorithm 6: BnB_MCS($G'_R, \hat{G}^*_R, \hat{g}^*, BE$)

Input: G'_R: current partial solution; $\hat{G}^*_R$: current best solution; $\hat{g}^*$: current best objective value; BE: current banned edges

```
1  A_mc ← DetectMCS(G'_R) ;
2  if A_mc = ∅ then
3      ĝ' ←ComputeObj(G'_R) ;
4      if ĝ' < ĝ* then
5          ĝ* ← ĝ';
6          Ĝ*_R ← G'_R;
7      return;
8  else
9      Res^L(A_mc) ←GetRankedResolvers(A_mc) ;
10     foreach (a_i, a_j) ∈ Res^L(A_mc) do
11         if (a_i, a_j) ∈ BE or (a_i, a_j) ∈ Tr(G'_R) or (a_j, a_i) ∈ Tr(G'_R) then
12             continue;
13         if ComputeLB_MCS(G'_R, a_i, a_j)< ĝ* then
14             G'_R ← (A_p, E(G'_R) ∪ {a_i, a_j});
15             R^L ←GetRankedResources(a_i, a_j);
16             for r_k ∈ R^L do
17                 if propagateLB(a_i, a_j, k)=true then
18                     BnB_MCS(G'_R, Ĝ*_R, ĝ*, BE);
19                     Restore();
20             G'_R ← (A_p, E(G'_R) \ {(a_i, a_j)});
21         BE ← BE ∪ {(a_i, a_j)};
22         if propagateUB(a_i, a_j)=true then
23             BnB_MCS(G'_R, Ĝ*_R, ĝ*, BE);
24             Restore();
25         BE ← BE \ {(a_i, a_j)};
26 return;
```

4.5.3 Constraint Propagation

In this section, we present our constraint propagation method in detail. For single resource problems, (Leus and Herroelen, 2004) proposes to maintain the flow bound consistency by conducting constraint propagation on the *remainder network* $G_{RD} = (A_p, E_p \cup E_{RD})$, where $E_{RD} = \{(a_i, a_j) \in FS | f_{ij}^U > 0\}$ is the set of edges not banned by the current branching decisions. For $(a_i, a_j) \in E(G_{RD})$, let $OT_{ij} = \{(a_i, a_l) \in E(G_{RD}) | l \neq j\}$ and $IN_{ij} = \{(a_l, a_j) \in E(G_{RD}) | l \neq i\}$ be the set of other edges in $E(G_{RD})$ that starts from a_i and ends at a_j, respectively. Since an AON-flow Network must satisfy inflow and outflow balance, the bounds of f_{ij} can be tightened using the following equations:

$$f_{ij}^L = \max \left\{ f_{ij}^L, b_i - \sum_{(a_i,a_l) \in OT_{ij}} f_{il}^U, b_j - \sum_{(a_l,a_j) \in IN_{ij}} f_{jl}^U \right\} \tag{4.16}$$

$$f_{ij}^U = \min \left\{ f_{ij}^U, b_i - \sum_{(a_i,a_l) \in OT_{ij}} f_{il}^L, b_j - \sum_{(a_l,a_j) \in IN_{ij}} f_{jl}^L \right\} \tag{4.17}$$

Consistency can be achieved by updating bounds for all edges in $E(G_{RD})$ till no bound changes. The network G_{RD}^T transformed from G_{RD} using the procedure in Section 4.4.1 is also used for detecting infeasibility in (Leus and Herroelen, 2004). If $f(G_{RD}^T) < f_{max}$, then clearly the current branching decisions cannot lead to any AON-flow Network, hence no POS can be generated according to Proposition 4.3.

For our problem with multiple resources, we maintain the flow bounds independently for each r_k based on Equations (4.16) and (4.17). The branching decisions on resources in Algorithm 6 enable the independent bound updates: when an edge (a_i, a_j) is included, f_{ijk}^L of a chosen r_k changes from 0 to 1 which makes the positive flow condition satisfied, and function propagateLB only maintains consistency for r_k; when (a_i, a_j) is banned, function propagateUB maintains consistency for all resources by setting f_{ijk}^U to 0 (so as f_{ijk}^L) for all r_k and propagating to other bounds. If any bound infeasibility (i.e. $f_{ijk}^U < f_{ijk}^L$) is detected during propagation, a false value is returned to signal the algorithm for backtracking. In addition to the early detection of infeasibility, another benefit of constraint propagation is that it may *imply* that certain edges $(a_i, a_j) \notin E(G_p)$ should be included (if $sum_{ij}^L > 0$) or banned (if $sum_{ij}^U = 0$).

If the flow bounds are updated successfully, propagateLB and propagateUB try to detect flow infeasibility. For each r_k, we maintain the transformed network $G^T_{RD}(k)$ and $G'^T_R(k)$ for the current partial solution G'_R and the the remainder network G_{RD}, and try to maximize flows in $G^T_{RD}(k)$ and $G'^T_R(k)$ for the resource r_k affected by constraint propagation. If $f(G^T_{RD}(k)) < f^k_{max}$ or $f(G'^T_R(k)) < f^k_{max}$, then according to Proposition 4.3, the current branching decisions cannot lead to any POS and a false value is returned to signal backtracking.

4.5.4 Heuristics for CS Reduction and Resolver Selection

In (Lombardi and Milano, 2012), the reduction of CS to MCS and resolver selection is based on the so-called preserved space heuristic designed in (Laborie, 2005), which estimates the amount of searching space left after adding a resolver. However, this heuristic is designed for deterministic RCPSP hence is not applicable to our problem due to the existence of multiple samples and time-dependent durations. Below we design a heuristic that evaluates resolvers from the perspective of resource constraints.

Essentially, by adding edges to a partial solution G'_R, we wish to increase the maximum flow in each $G'^T_R(k)$ to f^k_{max} so that a POS is obtained. Note that when $f(G'^T_R(k)) = f^k_{max}$ for all $r_k \in R$, the MCS detection function returns an empty set (Line 1 of Algorithm 6) since all resource conflicts have been resolved. Hence, we prefer the edge that can bring the largest increment for each $f(G'^T_R(k))$ so that a POS is reached as early as possible. Here we design a heuristic *Resource Score* to estimate the contribution that an eligible edge (a_i, a_j) could have for reaching a POS as follows:

$$RS(a_i, a_j) = \sum_{r_k \in R} \left\{ RS_k(a_i, a_j) = \frac{f^{RD}_{ijk}}{f^k_{max} - f(G'^T_R(k))} \right\}, \tag{4.18}$$

where $RS_k(a_i, a_j)$ is a normalized estimate for the contribution of (a_i, a_j) to resource r_k, with the nominator f^{RD}_{ijk} being the flow for r_k on edge (a_i, a_j) in the remainder network G_{RD} and the denominator being the current flow gap for G'^T_R to reach f^k_{max}.

Based on the resource score heuristic, we use a greedy procedure to reduce a CS to MCS in function DetectMCS (Line 1 of Algorithm 6). For a CS A_c, we define its resource score as the summation of the resource scores of all its activity pairs, i.e. $RS(A_c) = \sum_{(a_i,a_j)\in Res(A_c)} RS(a_i, a_j)$. We aim at obtaining a MCS $A_{mc} \subseteq A_c$ with

the highest resource score. Therefore, DetectMCS employs the following procedure to select a MCS: 1) for each $r_k \in R$, detect a CS A_c^k and reduce it to a MCS A_{mc}^k using a greedy procedure iteratively removes an activity from A_c^k that causes the smallest reduction in $RS(A_c^k)$ until a MCS is obtained; 2) return the A_{mc}^k with the maximum $RS(A_{mc}^k)$. Similar to (Lombardi and Milano, 2012), the greedy procedure of reducing a CS A_c^k to an MCS A_{mc}^k has a complexity of $\mathcal{O}(|A_c^k|^2)$.

Resource score is also used in ranking resolvers and resources for branching. More specifically, function GetRankedResolvers in Line 9 of Algorithm 6 ranks the resolvers in the descending order of their resource score values. Function GetRankedResources in Line 15 of Algorithm 6 ranks all resources also in the descending order, based on the values of $RS_k(a_i, a_j)$ of each resource r_k. For an MCS A_{mc}^k, the number of possible resolvers is $|A_{mc}^k|(|A_{mc}^k - 1|)$, therefore the resolver selection procedure has a complexity of $\mathcal{O}(|A_{mc}^k|^2)$.

4.5.5 Lower Bound

Here we design an admissible lower bound for `BnB-MCS` following the similar idea in Section 4.4.4, based on Equation (4.13) and Observation 4.2. Specifically, in the ComputeLB_MCS function of Algorithm 6 (Line 13), given a partial solution G'_R and a resolver (a_i, a_j), we first generate the new solution $\bar{G}'_R$ by including (a_i, a_j) to G'_R, then propagate it on each sample to obtain a lower bound for the makespan of using $\bar{G}'_R$, and finally take the average value as the lower bound of $\hat{g}$.

4.6 Experimental Results

In this section, we conduct a series of experiments to examine the performance of our algorithms on benchmark problem instances and different distributions from real-world data and literature. In Section 4.6.1, we first describe the general settings of our experiments. Then we examine different configurations of our algorithms and analyze the impact of different problem parameters in Section 4.6.2, on uncertainty models with both the time-dependent component $\boldsymbol{X}$ and the time-independent component $\boldsymbol{Y}$. Finally, in Section 4.6.3 and 4.6.4, we report the results on uncertainty models with only component $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively.

4.6.1 Experiment Setting

The RCPSP instances used in our experiments are generated using a widely used benchmark problem generator *RanGen2* (Vanhoucke et al., 2008). Five parameters are required to generate an instance, namely number of activities N, number of resources K, order strength (OS), resource factor (RF) and resource-constrainedness (RC). The values of OS, RF and RC are all chosen from $[0, 1]$. OS specifies the structure of the project network G, and a higher OS value indicates that G has more precedence constraints. RF and RC are used to specify the resource utilization status. In an instance with a higher RF value, more activities will have non-zero resource requirements b_i^k. On the other hand, a higher RC value specifies an instance where activities tend to require more resources (i.e. b_i^k is closer to C_k). More details of these parameters can be found in (Vanhoucke et al., 2008). We generate two sets of instances where the values of N and K are chosen from $\{10, 20, 30\}$ and $\{1, 2, 3\}$, respectively, but the values of OS, RF and RC are chosen from different ranges. Specifically, in Set1, the values of OS, RF and RC are chosen from {0.2,0.7} to represent the "low" and "high" level, while in Set2 we set OS$\in \{0.2, 0.4\}$, RF$\in \{0.7, 0.9\}$ and RC$\in \{0.2, 0.4\}$ to have more focused experiments since our approaches tend to show better performance on instances with lower OS, higher RF and lower RC. For each parameter combination, a subset with 10 instances are generated, therefore Set1 and Set2 contain 720 instances each. The duration of each activity a_i in these instances is an integer in $d_i^0 \in [1, 10]$.

To model the duration uncertainty $\boldsymbol{U}$, we need to model its two components, i.e. the time-dependent workability uncertainty $\boldsymbol{X}$ and the time-independent duration uncertainty $\boldsymbol{Y}$, respectively. Here we model $\boldsymbol{X}$ using a distribution dataset collected from a real-world aero engine testing project. As shown in Table 4.1 and visualized in Figure 4.8, this dataset describes the Probability of Workability (POW) of four types of activities in each month of a year. In our experiments, we assume that the scheduling horizon starts from the first date of a year. To obtain a sample of $\boldsymbol{X}$, we conduct random sampling for each activity type on each time slot of the horizon according to the corresponding POW value to determine the workability x_{zt}. For each activity in the generated RCPSP instances, we randomly assign a type $z \in \{1, 2, 3, 4\}$. In addition, except the experiments in Section 4.6.4, we increase the deterministic activity durations d_i^0 of each instance to elongate the critical path

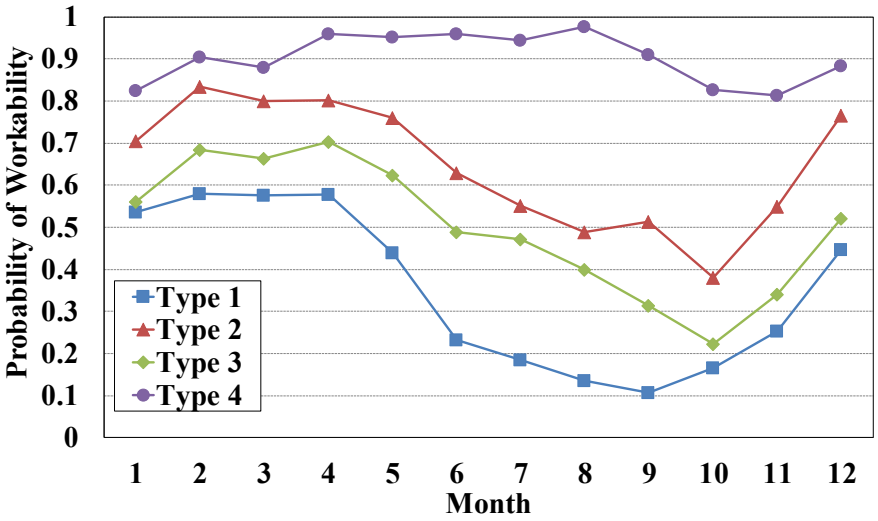


Figure 4.8: The Trend of Monthly POW

Table 4.1: Monthly POW Data

Month	1	2	3	4	5	6	7	8	9	10	11	12
Type 1	0.536	0.579	0.576	0.579	0.44	0.231	0.184	0.136	0.107	0.165	0.253	0.447
Type 2	0.704	0.833	0.8	0.802	0.76	0.628	0.552	0.488	0.512	0.38	0.549	0.766
Type 3	0.56	0.684	0.664	0.702	0.624	0.488	0.472	0.4	0.314	0.223	0.341	0.521
Type 4	0.824	0.904	0.88	0.959	0.952	0.959	0.944	0.976	0.909	0.826	0.813	0.883

length to a random integer value in [200, 300], such that most of the POW data can be covered. To model the time-independent component $\boldsymbol{Y}$, we use two distributions from the literature: 1) a normal distribution $\boldsymbol{Y}_i \sim \mathcal{N}(d_i^0, \sigma^2)$ with $\sigma = d_i^0 \times 0.5$, which is used in (Beck and Wilson, 2007); 2) an exponential distribution $\boldsymbol{Y}_i \sim Exp(1/d_i^0)$, which is used in (Creemers, 2015; Leus and Herroelen, 2004).

Since existing approaches cannot handle the time-dependent uncertainty, in most part of this section, we compare the quality of the solutions generated by our approaches with the ones given by general-purpose POS generation approaches. We call them general-purpose approaches because they directly generate POS for a given deterministic RCPSP instance, without the need of any probabilistic knowledge about the uncertainty. Therefore they can be applied to any type of uncertainty and is comparable in our experiments. Specifically, we implement three general-purpose approaches as benchmark algorithms:

- `ESTA-Iter`: this algorithm first generates a start-time schedule for the deter-

ministic problem, then transforms it using a chaining procedure to obtain a POS (Policella et al., 2004). Here we implement the iterative chaining procedure (100 iterations) in (Policella et al., 2009) to generate POS.

- `EBA-Minflow`: this algorithm shares similarities with our algorithm `BnB-MCS`, which also generates a POS by iteratively detecting and resolving MCS (Policella et al., 2004). However, the resource conflicts are resolved greedily, i.e. the precedence relations are added without backtracking. Here we implement this algorithm with the state-of-the-art MCS detection method in (Lombardi and Milano, 2012).

- `Artigues03`: this algorithm is proposed in (Artigues et al., 2003), which generates a feasible AON-flow Network (which is also a POS) based on the traditional priority-rule based schedule generation scheme.

In Section 4.6.4, which presents our results on uncertainty models with only component $\boldsymbol{Y}$, we also compare our algorithms with state-of-the-art solver for time-independent uncertain durations in (Creemers, 2015). We denote this approach as `Creemers15`, and more details will be given in Section 4.6.4. All the algorithms are implemented using JAVA[5], and run on an Intel Xeon Workstation (3.5GHz, 16GB). The CPU time of our branch-and-bound algorithms are limited to 300 seconds. If the optimal solution is not found, we use the best solution returned during searching. Since the expected makespan defined in Equation (4.5) is intractable to compute, we use Monte Carlo simulation to estimate the real objective g by $\hat{g}_{Q_s}(G_R)$, which is the value of the sample average function in Equation (4.6) on a set of Q_s testing samples. As suggested in (Kleywegt et al., 2002), this is a reliable way to estimate the expected value when the number of testing samples is large. Here we set $Q_s = 2000$ as in (Kleywegt et al., 2002).

4.6.2 Results on Models with Both Components

In this section, we conduct experiments on uncertainty models with both components $\boldsymbol{X}$ and $\boldsymbol{Y}$. Specifically, $\boldsymbol{X}$ is modeled using the dataset in Table 4.1, and $\boldsymbol{Y}$ is modeled using the normal distributions $\mathcal{N}(d_i^0, \sigma^2)$. We first examine the impact of different algorithm configurations, including sample size (Section 4.6.2.1)

[5]For `Creemers15`, we use the program from http://www.stefancreemers.be/software.php.

and branching heuristics (Section 4.6.2.2). Next we examine the impact of problem parameters on our algorithms in Section 4.6.2.3. Finally, we compare the quality of solutions given by our algorithms with those given by the benchmarking algorithms in Section 4.6.2.4.

4.6.2.1 Impact of Sample Size

We first examine the impact of sample size Q, which is an important parameter for SAA based approaches. Intuitively, a SAA problem with larger number of samples could produce solutions with higher quality, but requires longer computation time due to the increase of problem size. For SAA based approaches, the solution quality is often evaluated using the *optimality gap* proposed in (Kleywegt et al., 2002). To compute the optimality gap ρ for a given Q, first we replicate the SAA process by solving Q_{rep} SAA problems independently, each with its own sample set $\boldsymbol{u}_\eta$, $\eta \in \{1, ..., Q_{rep}\}$. Let G_R^η be the solution of each replication, and G_R^{rep} be the solution with the minimum value of $\hat{g}(G_R^\eta)$ on $\boldsymbol{u}_\eta$. Next, we generate $Q_s = 2000$ samples, and compute SAA value of G_R^{rep} as $\hat{g}_{Q_s}(G_R^{rep})$. Then the optimality gap value can be computed as:

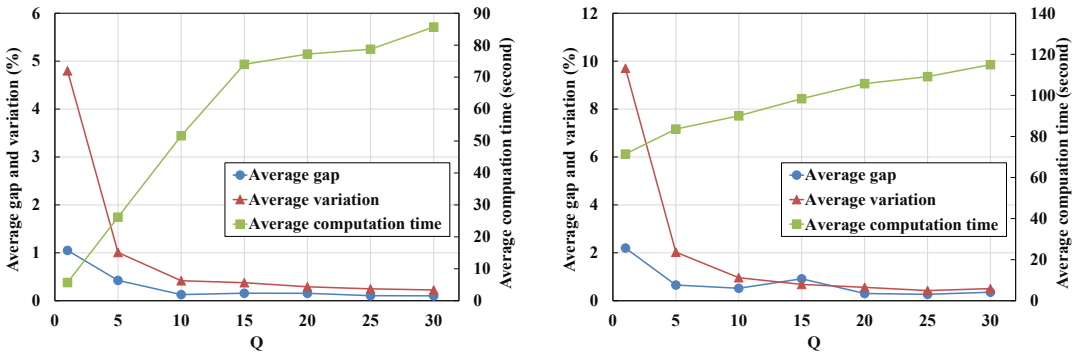
$$\rho = \left| \hat{g}_{Q_s}(G_R^{rep}) - \frac{1}{Q_{rep}} \sum_{\eta=1}^{Q_{rep}} \hat{g}(G_R^\eta) \right|. \tag{4.19}$$

Furthermore, we can estimate the variance of ρ as:

$$Var_\rho = \frac{Var_{Q_s}}{Q_s} + \frac{Var_{Q_{rep}}}{Q_{rep}}, \tag{4.20}$$

where Var_{Q_s} and $Var_{Q_{rep}}$ are the variance of the SAA values in Q_s times of simulations and Q_{rep} times of SAA replications, respectively. According to (Kleywegt et al., 2002), the lower value of ρ and Var_ρ, the higher quality of the produced solution.

Following (Kleywegt et al., 2002), we set the number of replications Q_{rep} to 20 in the experiments. The values of ρ and Var_ρ are normalized by the estimated objective value $\hat{g}_{Q_s}(G_R^{rep})$. For the purpose of clarity and brevity, we report the results on two representative instance subsets, each with 10 instances. Figure 4.9(a) shows the results for `BnB-Flow`, where the average normalized ρ and Var_ρ for a instance subset with 10 activities and 3 resources are plotted, along with the average computation time. Figure 4.9(b) shows the same curves for `BnB-MCS` on a subset



(a) BnB-Flow (b) BnB-MCS

Figure 4.9: Impact of Sample Size

with 20 activities and 3 resources. As shown in these figures, there exists a clear trade-off effect between the solution quality and computational cost. In general, the values of ρ and Var_ρ decrease with the increase of Q, and become stable when $Q \geq 20$. In the following experiments, we set $Q = 20$ according to this observation.

4.6.2.2 Impact of Algorithm Configurations

In this section, we study the performance of different algorithm configurations. First, we examine the performance of different branching heuristics we designed in Section 4.4.5 for BnB-Flow. The combination of these heuristics yields four possible configurations of Algorithm 4, including LFT+AEST, LFT+MLP, MTS+AEST, and MTS+MLP. In this section, we conduct experiments on the 240 instances from Set1 with $N = 20$ to examine the performance of these four configurations. Specifically, when one heuristic is used, the other one for the same branching level is used for tie-breaking. We classify the instances according to the four parameters K, OS, RF and RC, and report the results in Table 4.2. As shown in the table, LFT+AEST tends to give the best performance among all configurations. This is probably because they are more "focused" on evaluating the branching alternatives from the time aspect, which is in accord with the SAA objective function. On the other hand, their counterparts (i.e. MTS and MLP) are more focused on the graph characteristics of the solution. In the remaining experiments, we use LFT+AEST as the configuration for BnB-Flow.

Next, we examine the performance of the constraint propagation module in BnB-MCS. We run BnB-MCS with and without constraint propagation on the 240 in-

Table 4.2: Comparison of Branching Heuristics

Instance	LFT+AEST				LFT+MLP			
group	Best[1]	First[2]	Time[3]	PTO[4]	Best	First	Time	PTO
K=1	**624.33**	638.88	92.65	30	627.36	678.08	92.68	30
K=2	**682.65**	**694.07**	**120.41**	**40**	688.15	744.35	120.58	40
K=3	**721.44**	**733.87**	147.7	**48.75**	753.81	798.8	151.41	50
OS=0.2	**895.71**	**914.19**	**150.58**	**50**	920.58	1006.39	153.26	50.83
OS=0.7	**456.57**	**463.69**	89.93	29.17	458.96	474.43	89.85	29.17
RF=0.2	516.34	523.08	57.74	**19.17**	516.41	545.01	57.75	19.17
RF=0.7	**835.94**	**854.8**	**182.77**	**60**	863.13	935.81	185.36	60.83
RC=0.2	602.63	606.35	**33.23**	**10.83**	623.51	696.13	35.82	11.67
RC=0.7	**749.65**	**771.53**	207.28	68.33	756.04	784.69	207.29	68.33
Instance	MTS+AEST				MTS+MLP			
group	Best	First	Time	PTO	Best	First	Time	PTO
K=1	626.72	**636.52**	92.89	30	627.78	680.62	**89.14**	**28.75**
K=2	692.5	702.47	131.69	43.75	683.59	725.94	127.93	42.5
K=3	730.95	736.32	**146.98**	48.75	738.51	788.39	154.55	51.25
OS=0.2	908.31	918.78	165.66	55	908.54	989.36	165.65	55
OS=0.7	458.47	464.77	**82.04**	**26.67**	458.04	473.94	82.09	26.67
RF=0.2	**514.51**	**521.68**	**57.72**	19.17	516.39	537.48	57.73	19.17
RF=0.7	852.27	861.87	189.98	62.5	850.2	925.82	190.02	62.5
RC=0.2	**595.5**	**597.99**	42.82	14.17	613.86	688.38	42.83	14.17
RC=0.7	771.28	785.55	**204.88**	**67.5**	752.73	774.92	204.92	67.5

[1] The average of the best objective values upon termination.

[2] The average of the first objective values found in searching.

[3] The average computation time (in seconds).

[4] The percentage (%) of time-out instances.

Table 4.3: Effectiveness of Constraint Propagation in BnB-MCS

Instance	Without CP			With CP		
group	Best	Time	PTO	Best	Time	PTO
K=1	617.71	76.54	0.25	**611.34**	**53.23**	**0.18**
K=2	682.53	94.79	0.31	**662.08**	**81.61**	**0.25**
K=3	715.94	123.99	0.4	**708.19**	**110.25**	**0.36**
OS=0.2	891.14	145.42	0.48	**868.5**	**124.12**	**0.41**
OS=0.7	452.98	51.46	0.17	**452.57**	**39.28**	**0.12**
RF=0.2	507.62	10.08	0.03	**507.43**	**0.79**	**0**
RF=0.7	836.5	186.8	0.62	**813.65**	**162.6**	**0.53**
RC=0.2	**597.01**	60.2	0.2	598.37	**48.08**	**0.16**
RC=0.7	747.11	136.68	0.44	**722.7**	**115.32**	**0.37**

stances from Set1 with $N = 20$, and summarize the results in Table 4.3. As shown in this table, results with constraint propagation are better in almost all instance groups. While the solution quality is close, algorithm with constraint propagation shows significantly better computational efficiency, with 17% less average time (81.7 versus 98.4 seconds) and 18% less time-out instances (63 versus 77). In the remaining experiments, we will execute BnB-MCS with constraint propagation by default.

Finally, we examine the objective values of the first feasible solutions returned by BnB-Flow and BnB-MCS. We run the experiments on 720 instances in Set1, and report the average objective value of the first feasible solutions in Table 4.4, classified by the problem parameters. As shown in the table, the first solutions returned by BnB-Flow tend to have higher quality (on average 9% improvement). An intuitive explanation is that the resource conflict detection in BnB-MCS focuses on activities with the tightest resource contention, therefore resolving the detected MCS may not lead to a high quality solution in terms of makespan. On the contrary, the constructive procedure in BnB-Flow is more likely to link each activity in the way that it can start as early as possible. In order to achieve better pruning with a higher quality solution in the early stage, in the following experiments we will initialize BnB-MCS using the first feasible solution G_R^{FI} found by BnB-Flow. In other words, BnB-MCS is invoked by calling BnB_MCS$(G_p, G_R^{FI}, \hat{g}(G_R^{FI}), \emptyset)$.

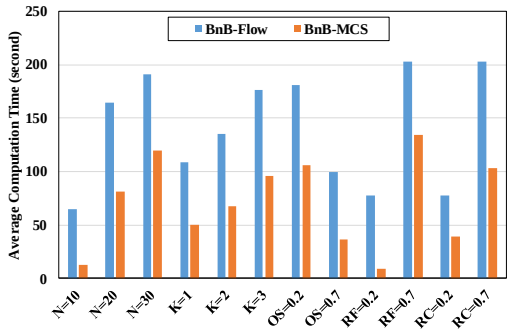
Table 4.4: Comparison of the First Feasible Solutions

Instance group	BnB-Flow	BnB-MCS	Diff(%)	Instance group	BnB-Flow	BnB-MCS	Diff(%)
N=10	**795.02**	808.79	1.73	OS=0.2	**958.26**	1083.2	13.04
N=20	**691.77**	768.69	11.12	OS=0.7	**497.79**	501.24	0.69
N=30	**697.29**	799.18	14.61	RF=0.2	**568.91**	580.32	2.01
K=1	**676.51**	725.22	7.2	RF=0.7	**887.15**	1004.12	13.19
K=2	**719.85**	778.09	8.09	RC=0.2	**616.22**	721.31	17.05
K=3	**787.72**	873.35	10.87	RC=0.7	**839.83**	863.13	2.77

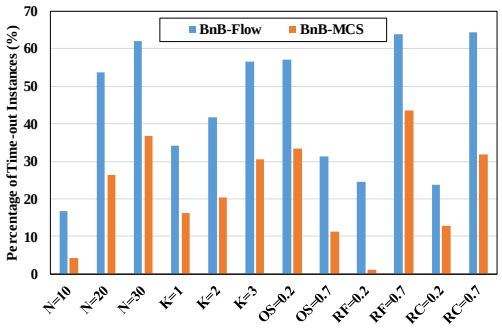
4.6.2.3 Impact of Problem Parameters

In this section, we examine the efficiency and solution quality of BnB-Flow and BnB-MCS, and analyze the impact of different problem parameters. We use the 720 instances in Set1 for these experiments. In general, BnB-Flow solves 402 (55.8%) instances optimally with an average computation time of 140.1 seconds. In comparison, BnB-MCS solves 559 (77.6%) instances optimally in 71.4 seconds on average. We believe the better scalability of the MCS-based algorithm is because the search space of POS can "summarize" that of AON-flow Network, since a POS could accommodate multiple feasible AON-flow Networks.

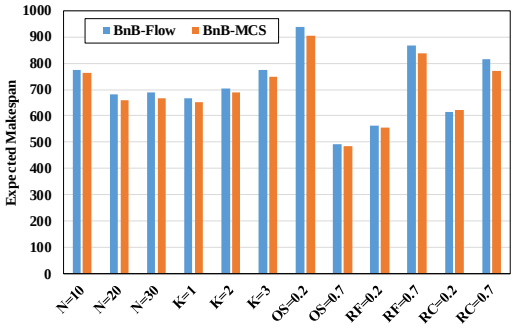
To study the impact of different problem parameters on the algorithm efficiency, we classify all instances in Set1 according to their parameters, and plot the average computation time and the percentage of time-out instances of the two algorithms in Figure 4.10(a) and 4.10(b), respectively. As shown in the figures, BnB-MCS shows better scalability for all instance groups. We also observe that the two algorithms share a common pattern for different parameter values, i.e. the hardness for solving an instance increases with N, K, RF and RC, but decreases with OS. Below we briefly analyze the rationale for this observation. Firstly, it is straightforward to see that the problem size grows with the increase of N and K. Secondly, recall that the OS value determines original AON network G, and an instance with a higher OS value has more precedence constraints. This will lead to a smaller search space for the two algorithms due to a) smaller number of branching alternatives in the activity level of BnB-Flow, and b) smaller number of MCS needed to be resolved by BnB-MCS. On the contrary, for the two resource-related parameters RF and RC, a



(a) Average Computation Time



(b) Average Percentage of Time-out Instances



(c) Average Expected Makespan

Figure 4.10: Impact of Problem Parameters on Algorithm Performance

higher value indicates a larger search space for the two algorithms, since a) more feasible links exist in the link level of `BnB-Flow`, and b) more activity combinations satisfy the conditions of MCS and need to be resolved by `BnB-MCS`.

In Figure 4.10(c), we plot the average objective values of the solutions produced by `BnB-Flow` and `BnB-MCS` for instance groups classified by different problem parameters. As shown, `BnB-MCS` tends to find better solutions than `BnB-Flow`. On the other hand, the difference is relatively small, which shows that `BnB-Flow` can find high-quality solutions even if the search is not exhausted. In fact, `BnB-Flow` returns the optimal solutions for 66 instances in the 157 ones closed by `BnB-MCS` but remain open for `BnB-Flow`.

4.6.2.4 Comparison with other Approaches

In this section, we compare the quality of solutions produced by our approaches with the ones generated by the benchmark algorithms, i.e. `ESTA-Iter`, `EBA`, and `Artigues03`. We first report and analyze the results on instances from Set1, which is

Table 4.5: Quality of Solutions on Models with Both Components - Set1

Instance group	BnB-Flow AvgObj	BnB-MCS AvgObj	ESA-Iter AvgObj	ESA-Iter Diff(%)[1]	EBA-Minflow AvgObj	EBA-Minflow Diff(%)	Artigues03 AvgObj	Artigues03 Diff(%)
N=10	772.66	**762.16**	793.21	4.07	804.31	5.53	808.06	6.02
N=20	683.39	**660.54**	705.18	6.76	743.33	12.53	724.32	9.66
N=30	689.72	**665.68**	709.22	6.54	782.34	17.52	727.3	9.26
K=1	665.4	**651.03**	669.69	2.87	704.75	8.25	693.9	6.58
K=2	704.9	**689.26**	726.91	5.46	767.59	11.36	745.3	8.13
K=3	775.46	**748.08**	811.01	8.41	857.63	14.64	820.48	9.68
OS=0.2	939.63	**905.92**	979.61	8.13	1059.12	16.91	1005.86	11.03
OS=0.7	490.88	**486.33**	492.13	1.19	494.19	1.62	500.59	2.93
RF=0.2	562.43	**554.03**	565.6	2.09	564.93	1.97	570.62	3
RF=0.7	868.08	**838.22**	906.14	8.1	988.38	17.91	935.83	11.64
RC=0.2	**615.58**	620.49	644.39	4.68	727.3	18.15	663.2	7.74
RC=0.7	814.93	**771.76**	827.34	7.2	826.01	7.03	843.25	9.26

[1] The difference (%) from the best value given by BnB-Flow and BnB-MCS.

listed in Table 4.5. As shown in Table 4.5, the results of BnB-MCS are the best among all instance groups, which clearly shows the advantage of incorporating the stochastic knowledge in the proactive scheduling problem. We also observe that EBA-Minflow performs worse than ESTA-Iter and Artigues03. A possible reason is that EBA-Minflow focuses more on the resource conflict detection and removing, but gives little attention to the precedence constraints between activities. On the contrary, ESTA-Iter explicitly considers minimizing the "dependencies" between activities (i.e. reducing the edges in POS).

Another interesting observation from Table 4.5 is that the improvement of our approach tends to be lower when the instances have higher OS, lower RF and higher RC. Here we give an intuitive explanation for this observation. For instances with higher OS values, the original project graph G is denser since more precedence constraints exist in E. In this case, a majority of edges in the final solution belong to E. For instances with lower RC values, the lower resource requirements of activities result in a relatively small number of additional edges in the final solution. For instances with higher RC value, the smaller improvement may result from the larger search spaces, in which our algorithms cannot return high quality solutions within

Table 4.6: Quality of Solutions on Models with Both Components - Set2

Instance	BnB-Flow	BnB-MCS	ESA-Iter		EBA-Minflow		Artigues03	
group	AvgObj	AvgObj	AvgObj	Diff(%)	AvgObj	Diff(%)	AvgObj	Diff(%)
N=10	1009.73	**965.27**	1096.28	13.57	1176.56	21.89	1122.76	16.32
N=20	1044.3	**1014.6**	1168.89	15.21	1354.62	33.51	1247.05	22.91
N=30	1094.51	**1082.56**	1223.02	12.98	1532.78	41.59	1275.93	17.86
K=1	955.17	**934.31**	1050.44	12.43	1252.15	34.02	1100.54	17.79
K=2	1047.7	**1021.09**	1166.48	14.24	1352.2	32.43	1221.43	19.62
K=3	1145.68	**1107.03**	1271.26	14.84	1459.61	31.85	1323.76	19.58
OS=0.2	1276.89	**1246.59**	1504.11	20.66	1819.71	45.97	1533.23	22.99
OS=0.5	822.14	**795.02**	821.35	3.31	889.59	11.9	897.27	12.86
RF=0.7	977.05	**956.29**	1068.64	11.75	1222.25	27.81	1118.36	16.95
RF=0.9	1121.98	**1085.32**	1256.82	15.8	1487.06	37.02	1312.13	20.9
RC=0.2	**852.82**	854.07	932.21	9.31	1163.81	36.47	1001.6	17.45
RC=0.4	1246.21	**1187.55**	1393.24	17.32	1545.49	30.14	1428.89	20.32

the time limit. To further study the performance of our algorithms on lower OS, higher RF and lower RC, we conduct experiments on the 720 instances from Set2. In this test set, BnB-Flow and BnB-MCS closes 133 and 324 instances with the average computation time of 248.1 and 171.5 seconds, respectively. The results are summarized in Table 4.6, which shows a more prominent improvement.

4.6.3 Results on Models with Component $\boldsymbol{X}$

In this section, we summarize the experiments on uncertainty models with only component $\boldsymbol{X}$ modeled by the dataset in Table 4.1, while component $\boldsymbol{Y}$ is deterministic (i.e. $Pr(\boldsymbol{Y}_i = d_i^0) = 1$ for all i). We report the results on Set1 and Set2 in Table 4.7 and 4.8, respectively. Compared to the corresponding values in Table 4.5 and 4.6, the expected makespan values in these two tables are smaller. This is reasonable since now only one uncertainty source exists. We also have similar observations as the ones in Section 4.6.2.4, which can be explained by similar rationale.

4.6.4 Results on Models with Component $\boldsymbol{Y}$

In this section, we report the experiments on uncertainty models that only consist of component $\boldsymbol{Y}$, i.e. $Pr(\boldsymbol{X}_{zt} = 1)$ for all z and t. We also restore the deterministic

Table 4.7: Quality of Solutions on Models with Component $\boldsymbol{X}$ - Set1

Instance	BnB-Flow	BnB-MCS	ESTA-Iter		EBA-Minflow		Artigues03	
group	AvgObj	AvgObj	AvgObj	Diff(%)	AvgObj	Diff(%)	AvgObj	Diff(%)
N=10	699.36	**697.27**	722.31	3.59	744.01	6.7	743.07	6.57
N=20	627.75	**614.06**	654.82	6.64	722.88	17.72	678.14	10.44
N=30	639.25	**618.23**	656.47	6.18	754.91	22.11	682.15	10.34
K=1	608.62	**595.24**	615.6	3.42	673.94	13.22	644.27	8.24
K=2	646.7	**638.34**	667.45	4.56	733.08	14.84	693.11	8.58
K=3	711.04	**695.99**	750.55	7.84	814.69	17.05	765.98	10.06
OS=0.2	842.39	**821.37**	887.06	8	1001.15	21.89	921.62	12.21
OS=0.7	468.51	**465.01**	468.67	0.79	478.81	2.97	480.62	3.36
RF=0.2	512.16	**503.17**	514.44	2.24	520.2	3.38	879.89	3.81
RF=0.7	798.75	**783.21**	841.29	7.41	961.58	22.77	522.35	12.34
RC=0.2	548.6	**548.27**	573.08	4.53	695.12	26.78	599.52	9.35
RC=0.7	762.31	**738.11**	782.65	6.03	786.17	6.51	802.72	8.75

Table 4.8: Quality of Solutions on Models with Component $\boldsymbol{X}$ - Set2

Instance	BnB-Flow	BnB-MCS	ESTA-Iter		EBA-Minflow		Artigues03	
group	AvgObj	AvgObj	AvgObj	Diff(%)	AvgObj	Diff(%)	AvgObj	Diff(%)
N=10	897.02	**878.16**	996.22	13.44	1127.67	28.41	1028.67	17.14
N=20	**941.3**	941.66	1079.82	14.72	1333.94	41.71	1170.69	24.37
N=30	992.41	**991.35**	1117.48	12.72	1513.08	52.63	1185.34	19.57
K=1	**852.64**	862.31	957.02	12.24	1213.79	42.36	1021.23	19.77
K=2	937.84	**931.03**	1070.7	15	1340.17	43.95	1131.93	21.58
K=3	1040.23	**1017.82**	1165.79	14.54	1420.72	39.58	1231.53	21
OS=0.2	**1119.1**	1130.08	1369.51	22.38	1754.99	56.82	1408.78	25.89
OS=0.5	768.05	**744.03**	759.5	2.08	894.8	20.26	847.68	13.93
RF=0.7	874.67	**870.4**	968.65	11.29	1210.94	39.12	1031.08	18.46
RF=0.9	1012.47	**1003.71**	1160.36	15.61	1438.86	43.35	1225.38	22.08
RC=0.2	750.16	**749.57**	826.36	10.24	1142.79	52.46	911.96	21.67
RC=0.4	1136.98	**1124.54**	1302.65	15.84	1507	34.01	1344.5	19.56

durations d_i^0 to the original values (i.e. integers in $[1, 10]$) since $\boldsymbol{X}$ is not considered here. In this case, the proactive scheduling problem in Equation (4.5) is reduced to the traditional stochastic RCPSP. As mentioned in Section 4.6.1, the current best approach for solving stochastic RCPSP with the objective of minimizing expected makespan is Creemers15, where the stochastic scheduling procedure is considered as a continuous time Markov Decision Process, and the optimal scheduling policy is found by dynamic programming technique (Creemers, 2015). When the activity duration follows exponential distribution, i.e. $\boldsymbol{Y}_i \sim Exp(1/d_i^0)$, the expected makespan returned by Creemers15 is the actual optimal value. Therefore, we conduct experiments on Set1 and Set2 with exponential distributions, and compare the solution qualities by computing the gap (%) of a solution's objective value given by our algorithms or benchmarks to the optimal expected makespan given by Creemers15. As we have mentioned in Section 2.3, the solution of Creemers15, i.e. elementary policy, represents a much larger solution space than POS, therefore it is expected that the expected makespan given by Creemers15 is lower than ours. However, Creemers15 is not anytime and can only terminate when the optimal expected makespan is found. In our experiments, Creemers15 solves 690 and 685 instances for Set1 and Set2, respectively, with a limitation of 16GB memory and 300 seconds running time. Below we only report the results for the instances solved by Creemers15.

The results are summarized in Table 4.9 and 4.10. On both Set1 and Set2, our algorithms consistently outperform the three benchmark algorithms. For Set1, our two algorithms can find solutions within 5% to the optimal expected makespan, while BnB-MCS tends to perform better than BnB-Flow. For Set2, the gaps become larger for all algorithms, and the results of BnB-Flow and BnB-MCS are close. The increasing of optimality gap on Set2 is probably because the parameter configuration for Set2 results in a larger policy space for Creemers15, which gives more possibility for finding an optimal policy that has a much better expected makespan than the optimal POS.

4.7 Conclusions

In this chapter, we study the problem of proactive scheduling with the objective of minimizing the expected makespan. Different from previous approaches, we al-

Table 4.9: Quality of Solutions on Models with Component $\boldsymbol{Y}$ - Set1

Instance group	BnB-Flow[1]	BnB-MCS	ESTA-Iter	EBA-Minflow	Artigues03
N=10	**0.43**	0.47	2.3	2.39	3.5
N=20	3.29	**2.41**	5.79	6.58	7.23
N=30	5.15	**4.1**	7.74	8.33	8.53
K=1	1.62	**1.43**	2.75	3.96	4.27
K=2	3.12	**2.39**	5.47	5.95	7.04
K=3	3.81	**2.9**	7.24	7.01	7.63
OS=0.2	4.75	**4.19**	8.49	10.06	10.01
OS=0.7	1.13	**0.47**	2.12	1.61	2.96
RF=0.2	0.75	**0.24**	1.73	1.13	1.69
RF=0.7	4.92	**4.19**	8.51	10.05	10.84
RC=0.2	**1.77**	1.87	5.4	6.38	5.99
RC=0.7	3.95	**2.62**	4.94	4.93	6.67

[1] The average gap (%) to the value given by Creemers15.

Table 4.10: Quality of Solutions on Models with Component $\boldsymbol{Y}$ - Set2

Instance group	BnB-Flow	BnB-MCS	ESTA-Iter	EBA-Minflow	Artigues03
N=10	5.78	**5.34**	11.88	13.44	14.22
N=20	**13.01**	13.27	23	25.15	29.62
N=30	**17.56**	17.96	24.56	23.21	31.25
K=1	**8.3**	9.28	15	20	19.93
K=2	12.43	**12.41**	19.99	19.55	25.51
K=3	14.7	**13.94**	23.62	21.82	28.6
OS=0.2	**15.07**	16.04	29.66	31.37	32.71
OS=0.5	8.92	**8.15**	10.46	10.62	17.49
RF=0.7	9.67	**9.61**	16.42	17.69	20.5
RF=0.9	**14.02**	14.18	22.73	23.26	28.94
RC=0.2	**7.01**	7.29	14.54	16.41	20.25
RC=0.4	16.76	**16.59**	24.7	24.61	29.27

low the activity duration uncertainty to be time-dependent, caused by the time-dependent workability uncertainty. We propose a stochastic optimization problem that can incorporate the traditional stochastic RCPSP model and the time-dependent workability uncertainty model at the same time. The resulting proactive problem is very challenging, since even evaluating a solution is computationally intractable. To tackle the hardness in solution evaluation, we approximate the problem based on SAA, which is a principled approximation scheme with convergence guarantee. We prove that the resulting SAA problem is still NP-hard, due to the combinatorial nature of RCPSP.

We then propose two branch-and-bound algorithms to solve the SAA problem optimally. The first algorithm uses a constructive approach to extend a partial temporal network with part of activities to a full feasible solution, by identifying precedence and resource feasible links. The second algorithm finds a feasible solution by iteratively detecting and removing possible resource conflicts, until a temporal network is proved to be conflict-free. By exploiting some properties of the SAA problem, we design several components for the branch-and-bound algorithms, including branching heuristics and lower bounds. To verify the performance of our algorithms, we conduct a series experiments on pure workability uncertainty, pure time-independent duration uncertainty, and mixture models with two uncertainty sources that are built from real-world dataset and common distributions used in the literature.

Chapter 5

Risk-Aware Proactive Scheduling via Conditional Value-at-Risk

The previous chapter studies the problem of proactive scheduling for centralized RCPSP, with the objective of minimizing the expected makespan. However, such a risk-neutral criterion may not be realistic when the actual project execution needs to be controlled by certain risk requirement. In this chapter, we study the proactive scheduling problem for RCPSP with the objective of minimizing the robust makespan, which is a risk-aware objective. State-of-the-art approaches for solving this problem rely on probabilistic constraint optimization, which leads to intractable Mixed Integer Linear Programs (MILP) that does not scale to large sets of samples, leading to unsatisfactory solution quality. We resolve this issue by conducting the risk-aware optimization via Conditional Value-at-Risk (CVaR), a coherent risk measure. Since previous CVaR optimization approaches are not applicable to combinatorial problems, we propose a general branch-and-bound framework for combinatorial CVaR optimization. By instantiating this framework with the two algorithms (`BnB-Flow` and `BnB-MCS`) we proposed in Chapter 4, we obtain efficient risk-aware algorithms that scales well to hundreds of samples with significantly better solution quality.

This chapter is structured as follows. We first introduce the optimization of two popular risk measures, i.e. Value-at-Risk (VaR) and CVaR. In Section 5.2, we formulate our risk-aware proactive problem based on CVaR. In Section 5.3, we present the general framework for combinatorial CVaR minimization, which is further instantiated in Section 5.4 to solve the proactive problem. Empirical evaluation of our

algorithms is provided in Section 5.5, followed by the conclusions in Section 5.6.

5.1 Preliminaries: Minimizing VaR and CVaR

In this section, we introduce the basic concepts of two widely used measures for risk management, Value-at-Risk (VaR) and Conditional Value-at-Risk (CVaR). Let x be a decision vector with domain $\mathcal{X}$, and $g = g(\mathrm{x}, \mathbf{y})$ be the loss function of x on a random vector $\mathbf{y}$. Given a confidence level $\beta \in (0, 1)$, the β-VaR of x is defined as $\zeta_\beta(\mathrm{x}) = \min\{\zeta | Pr(g \leq \zeta) \geq \beta\}$, which is the β quantile of the random loss g. The β-CVaR of x is defined as $\phi_\beta(x) = \mathbb{E}[g | g \geq \zeta_\beta(\mathrm{x})]$, which is the expected loss beyond β-VaR. For risk-aware settings, x with smaller VaR or CVaR is more preferable. Hence, the best decision x^* can be found by minimizing VaR or CVaR in $\mathcal{X}$.

In the theory of risk management, CVaR is believed to be a more realistic and desirable objective than VaR, mainly for two reasons. Firstly, CVaR is computationally more tractable than VaR since it is mathematically coherent. Also, CVaR is an upper bound for VaR (i.e. $\phi_\beta(\mathrm{x}) \geq \zeta_\beta(\mathrm{x})$ holds for any $\mathrm{x} \in \mathcal{X}$), and the decision with a smaller CVaR tends to have a smaller VaR too. Secondly, VaR only provides a bound for loss g but does not quantify the loss beyond that bound. In contrast, by definition, CVaR explicitly captures this using the conditional expectation. Detail discussion about the superiority of CVaR over VaR can be found in (Rockafellar and Uryasev, 2002).

The minimization of CVaR is often done by minimizing a function $F_\beta = F_\beta(\mathrm{x}, \omega)$ defined as follows:

$$F_\beta(\mathrm{x}, \omega) = \omega + \frac{1}{1-\beta}\mathbb{E}\{[g(\mathrm{x}, \mathbf{y}) - \omega]^+\}, \tag{5.1}$$

where ω is an additional real variable and $[\cdot]^+ = \max\{\cdot, 0\}$. It has been shown in (Rockafellar and Uryasev, 2002) that CVaR minimization in $\mathcal{X}$ has an equivalent form:

$$(\mathrm{x}^*, \omega^*) = \underset{(\mathrm{x},\omega)\in\mathcal{X}\times\mathbb{R}}{\operatorname{argmin}} F_\beta(\mathrm{x}, \omega), \tag{5.2}$$

where x^* minimizes CVaR. Since F_β has an expectation form, Sample Average Approximation (SAA) (Kleywegt et al., 2002) is immediately applicable to approximate

Problem (5.2), by optimizing $\hat{F}_\beta = \hat{F}_\beta(\mathrm{x}, \omega)$ defined below in the joint space $\mathcal{X} \times \mathbb{R}$:

$$\hat{F}_\beta(\mathrm{x}, \omega) = \omega + \frac{1}{(1-\beta)Q} \sum_{q=1}^{Q} [g(\mathrm{x}, \mathrm{y}^q) - \omega]^+, \tag{5.3}$$

where $(\mathrm{y}^1, ..., \mathrm{y}^Q)$ are Q samples independently drawn from $\mathbf{y}$. Guaranteed by the property of SAA, the optimal solution $(\hat{\mathrm{x}}^*, \hat{\omega}^*)$ is proven to converge to (x^*, ω^*) in Equation (5.2) exponentially fast with the increase of sample size Q.

5.2 CVaR based Proactive Scheduling

As mentioned, we aim at optimizing the α-robust makespan. Here we give the definition of this problem on the generalized uncertainty model $\boldsymbol{U} =< \boldsymbol{X}, \boldsymbol{Y} >$ we proposed in Section 4.2.1, following the problem definition in (Beck and Wilson, 2007) for job-shop problems. Given a risk parameter $\alpha \in (0, 1)$, a real value D is α-achievable for POS G_R if $Pr(MS(G_R, \boldsymbol{U}) \leq D) \geq 1-\alpha$, i.e. the probability that the random makespan exceeds D is bounded by α. Then the α-robust makespan of G_R is the minimum of all the α-achievable D, i.e. $D_\alpha(G_R) = \min\{D | Pr(MS(G_R, \boldsymbol{U}) \leq D) \geq 1-\alpha\}$. The proactive problem can be formulated as:

$$G_R^* = \underset{G_R \in \boldsymbol{G}_R}{\operatorname{argmin}} D_\alpha(G_R). \tag{5.4}$$

It is easy to verify that $D_\alpha(G_R)$ is the β-VaR of G_R on $\boldsymbol{U}$, with $\beta = 1-\alpha$ and the loss being $MS(G_R, \boldsymbol{U})$. Hence Problem (5.4) is equivalent to minimizing the β-VaR. Note that when the uncertainty model $\boldsymbol{U}$ only contains component $\boldsymbol{Y}$, i.e. there is no workability uncertainty, the problem in Equation (5.4) corresponds to the problems studied in (Fu et al., 2012, 2016).

The typical way for VaR minimization is to transform it into a chance constrained optimization problem (Hong et al., 2014). However, for Problem (5.4), this results in MILPs that are computationally prohibitive, as presented in (Varakantham et al., 2016; Fu et al., 2016). Therefore, we take a different approach here which optimizes CVaR instead of VaR. To be more specific, we optimize the approximate function $\hat{F}_\beta$ defined in Equation (5.3) on independent samples $\{u^1, ..., u^Q\}$ drawn from $\boldsymbol{U}$:

$$(\hat{G}_R^*, \hat{\omega}^*) = \underset{(G_R, \omega) \in \boldsymbol{G}_R \times \mathbb{R}}{\operatorname{argmin}} \hat{F}_\beta(G_R, \omega). \tag{5.5}$$

Then the optimal solution $(\hat{G}_R^*, \hat{\omega}^*)$ for the above problem converges to the actual CVaR minimizing solution (G_R^*, ω^*) exponentially fast with the increase of Q.

As mentioned in (Hong et al., 2014), most CVaR optimization approaches assume that the decision space $\mathcal{X}$ is linear. In this case, the optimization of $\hat{F}_\beta$ can be transformed to a linear program which can be solved efficiently. However, our problem does not comply with this assumption since the POS space $\boldsymbol{G}_R$ is combinatorial. Similar to Proposition 4.2, we can show that it is NP-hard to optimize $\hat{F}_\beta(G_R, \omega)$:

Proposition 5.1 *The SAA problem in Equation (5.5) is NP-hard.*

Proof. We first construct an RCPSP instance from a PIT instance with graph $G = (V, E)$, following the same procedure as in the proof of Proposition 4.2. Then we add one sample u with $x_{zt} = 1$ for all z and t, and $y_i = 1$ for all activity a_i, and obtain an instance of problem (5.5) with the objective $\hat{F}_\beta(G_R, \omega) = \omega + 1/(1-\beta)[MS(G_R, u) - \omega]^+$. We claim that this problem has a solution (G_R, ω) with $\hat{F}_\beta(G_R, \omega) \leq t$ if and only if the corresponding PIT instance has a solution. Firstly, if the PIT instance has a solution, then a feasible schedule S can be obtained immediately with $MS(S) \leq t$. Then on each resource unit, we sort the consuming activities in ascending order based on their start times in S, and create a POS G_R by adding precedence constraints for each activity and its immediate successor on each resource it consumed. Clearly $MS(G_R, u) \leq t$, hence we have a solution (G_R, ω) with $\omega = MS(G_R, u)$ and $\hat{F}_\beta(G_R, \omega) = MS(G_R, u) \leq t$. Secondly, if problem (5.5) has a solution (G_R, ω) with $\hat{F}_\beta(G_R, \omega) \leq t$, then we must have $MS(G_R, u) \leq t$. This is because if $MS(G_R, u) = t' > t$, then function $\hat{F}_\beta(\omega) = \omega + 1/(1-\beta)[t' - \omega]^+$ has an infimum t', indicating $\hat{F}_\beta(G_R, \omega) > t$ for any $\omega \in \mathbb{R}$. Hence schedule $S = S(G_R, u)$ has a makespan $MS(S) \leq t$, indicating the PIT instance has a solution. □

Therefore, linear approaches cannot be applied to solve the SAA problem in Equation (5.5). In the next section, we design a general branch-and-bound framework for combinatorial CVaR minimization.

5.3 A Branch-and-bound Framework for Combinatorial CVaR Minimization

In general, a branch-and-bound algorithm iteratively partitions the solution space into smaller pieces, and uses a bounding function to fathom searching in certain solution pieces. Here we aim at designing a branch-and-bound framework for minimizing $\hat{F}_\beta$ defined in Equation (5.3) in the solution space $\mathcal{X} \times \mathbb{R}$ where $\mathcal{X}$ is combinatorial. Our framework only partitions $\mathcal{X}$, since ω is an unbounded real variable which is relatively easy to optimize.

The core component of a branch-and-bound algorithm is the bounding function. Below we design a lower bounding function for minimizing $\hat{F}_\beta$. Given a subset of decisions $\mathcal{X}' \subseteq \mathcal{X}$, suppose we can lower bound the loss function g on each sample y^q for $\mathcal{X}'$, by calling a function $g_{LB}(\mathcal{X}', y^q)$. Next, we define an auxiliary function L_β as follows:

$$L_\beta(\mathcal{X}', \omega) = \omega + \frac{1}{(1-\beta)Q} \sum_{q=1}^{Q} [g_{LB}(\mathcal{X}', \mathrm{y}^q) - \omega]^+. \tag{5.6}$$

Then we can have the following conclusion:

Proposition 5.2 *Define $LB(\mathcal{X}')$ as follows:*

$$LB(\mathcal{X}') = \min_{\omega \in \mathbb{R}} L_\beta(\mathcal{X}', \omega), \tag{5.7}$$

then $LB(\mathcal{X}')$ is a lower bound for $\mathcal{X}'$.

Proof. For any decision $\mathrm{x} \in \mathcal{X}'$ and sample y^q, we have $g_{LB}(\mathcal{X}', \mathrm{y}^q) \leq g(\mathrm{x}, \mathrm{y}^q)$ since g_{LB} is a lower bound. Then for any $\omega \in \mathbb{R}$, we have $L_\beta(\mathcal{X}', \omega) \leq \hat{F}_\beta(\mathrm{x}, \omega)$. In other words, for any $\mathrm{x} \in \mathcal{X}'$, function L_β is pointwise smaller than or equal to $\hat{F}_\beta$ with respect to ω. Therefore the minimum value of L_β with respect to ω, i.e. $LB(\mathcal{X}')$, should satisfy $LB(\mathcal{X}') \leq \hat{F}_\beta(\mathrm{x}, \omega)$ for any $(\mathrm{x}, \omega) \in \mathcal{X}' \times \mathbb{R}$. □

Remark. We can also conclude that if g_{LB} is stronger, then $LB(\mathcal{X}')$ is also stronger which leads to more effective pruning. For any $\mathcal{X}'$ and y^q, if $g^1_{LB}(\mathcal{X}', \mathrm{y}^q) \leq g^2_{LB}(\mathcal{X}', \mathrm{y}^q)$, then the corresponding two functions L^1_β and L^2_β satisfy $L^1_\beta(\mathcal{X}', \omega) \leq L^2_\beta(\mathcal{X}', \omega)$ for any $\omega \in \mathbb{R}$. Therefore, $\min_{\omega \in \mathbb{R}} L^1_\beta(\mathcal{X}', \omega) \leq \min_{\omega \in \mathbb{R}} L^2_\beta(\mathcal{X}', \omega)$.

According to Proposition 5.2, the lower bound value can be computed in two steps: 1) compute the lower bound on each sample; 2) solve the optimization problem

(5.7). It is easy to verify that L_β is convex (though not differentiable) with respect to ω, therefore (5.7) is a univariate convex optimization problem, which can be solved by standard techniques such as the subgradient method. Nevertheless, we can show that Problem (5.7) can be solved more efficiently by simply ranking the sample lower bounds.

Proposition 5.3 *If Q sample lower bound values are ranked ascendingly as $g^1_{LB} \leq ... \leq g^Q_{LB}$, then $\omega^* = g^{q^*}_{LB}$ with $q^* = \lceil \beta Q \rceil$ solves Problem (5.7) optimally.*

Proof. The ranked sample lower bounds split $\mathbb{R}$ into a set of intervals $(-\infty, g^1_{LB}]$, ...,$[g^q_{LB}, g^{q+1}_{LB}]$,...,$[g^Q_{LB}, \infty)$. It is easy to verify that L_β is linear in each interval, and is continuous in $\mathbb{R}$. We can then write the derivative of L_β with respect to ω: when $\omega \leq g^1_{LB}$, $L'_\beta = -\beta/(1-\beta) < 0$; for any integer $q \in [1, Q-1]$, when $\omega \in [g^q_{LB}, g^{q+1}_{LB}]$, $L'_\beta = (q - \beta Q)/(Q - \beta Q)$; when $\omega \geq g^Q_{LB}$, $L'_\beta = 1 > 0$. Hence, along with the increase of ω in $\mathbb{R}$, L'_β increases from negative to positive. The smallest q that makes $L'_\beta \geq 0$ is $q^* = \lceil \beta Q \rceil$, meaning that L_β stops decreasing in $[g^{q^*}_{LB}, g^{q^*+1}_{LB}]$ and $\omega^* = g^{q^*}_{LB}$ is an optimal solution to Problem (5.7). □

Therefore, $LB(\mathcal{X}')$ can be obtained very easily after computing the sample lower bounds. With proper branching functions to partition the solution space $\mathcal{X}$, the branch-and-bound process can be executed correctly to find the optimal solution $(\hat{\mathrm{x}}^*, \hat{\omega}^*)$. Note that when a feasible decision $\mathrm{x}' \in \mathcal{X}$ is reached, a candidate (x', ω') for the optimal solution can be obtained by fixing the loss $g(\mathrm{x}', \mathrm{y}^q)$ in Equation (5.3) for each sample, and minimizing $\hat{F}_\beta$ with respect to ω following a similar ranking procedure as minimizing L_β. The sample losses can also be used to retrieve the (approximate) β-VaR of x' on the samples $(\mathrm{y}^1, ..., \mathrm{y}^Q)$, i.e. $\hat{\phi}_\beta(\mathrm{x}') = \max\{g(\mathrm{x}', \mathrm{y}^q) | g(\mathrm{x}', \mathrm{y}^q) \leq \omega'\}$ (Rockafellar and Uryasev, 2002). This framework is general and applicable for any combinatorial CVaR minimization problem, as long as the sample bounding function g_{LB} is available.

5.4 The Risk-Aware Proactive Algorithms

In this section, we instantiate our CVaR minimization framework to solve Problem (5.5) by adapting the two risk-neutral algorithms in Chapter 4, i.e. `BnB-Flow` and `BnB-MCS`. The branching decisions for partition the solution spaces are already

provided in Algorithm 4 and 6, therefore we only need to modify the computation of objective values and lower bounds.

For `BnB-Flow`, we first need to modify the objective computation function ComputeObj (Line 3 of Algorithm 4). This can be done by computing the makespan of the POS on each sample using Algorithm 3, then compute the objective value following the ranking procedure we mentioned in the last section. Next, we need to modify the two lower bound computation functions in the two branching levels, i.e. ComputeLB_A (Line 11 of Algorithm 4) and ComputeLB_L (Line 3 of Algorithm 5). Since the lower bound computation in our CVaR optimization framework only requires a sample bounding function, we can use the corresponding methods in Section 4.4.4 to compute the lower bound on each sample, then use the ranking procedure to compute the lower bound for the CVaR objective function $\hat{F}_\beta$.

The adaptation of `BnB-MCS` is similar to the modification of `BnB-Flow`. Firstly, we modify the ComputeObj function (Line 3 of Algorithm 6) for computing the CVaR objective. Then, for the lower bounding function ComputeLB_MCS (Line 13 of Algorithm 6), we use the method in Section 4.5.5 to compute the sample lower bounds. Note that when a POS G_R is reached, the algorithm can backtrack safely. Because for any POS G_R and G'_R with $E(G_R) \subseteq E(G'_R)$, $MS(G_R, u) \leq MS(G'_R, u)$ holds for any sample u, therefore $\min_{\omega \in \mathbb{R}} \hat{F}_\beta(G_R, \omega) \leq \hat{F}_\beta(G'_R, \omega)$ holds for any ω.

5.5 Experimental Results

In this section, we empirically evaluate our approach on benchmark instances, and compare with two state-of-the-art approaches `SORU-H` (Varakantham et al., 2016) and `BACCHUS` (Fu et al., 2016). `SORU-H` computes start-time schedule, while `BACCHUS` generates POS as our approach does. Our algorithms `BnB-Flow` and `BnB-MCS` are implemented in JAVA 1.8, while `SORU-H` and `BACCHUS` are coded using Java API for CPLEX 12.7.1. For `BnB-Flow`, we use LFT+AEST as the branching heuristics; for `BnB-MCS`, we execute the algorithm with constraint propagation. All algorithms run on an Intel Xeon E5 Workstation (3.5GHz, 16GB).

Though our algorithms are applicable to the generalized uncertain duration models in Section 4.2.1, we focus on the pure time-independent model (i.e., $Pr(\boldsymbol{X}_{zt} = 1) = 1$ for all z and t) in most part of this section since the benchmark approaches

can only solve the time-independent models. The RCPSP instances used in most of this section are the 720 instances from Set1 in Section 4.6, without elongation of the activity durations. Two distributions are used to model the uncertainty: 1) a normal distribution $\mathcal{N}(d_i^0, \sigma^2)$ with d_i^0 being the deterministic duration of a_i (an integer in [1, 10] in the generated instances) and $\sigma = 0.5$, which is used in (Fu et al., 2012; Varakantham et al., 2016); 2) an exponential distribution $Exp(1/d_i^0)$ used in (Creemers, 2015; Leus and Herroelen, 2004). The uncertainty level of Exp is higher than $\mathcal{N}$, since its squared coefficient of variance (SCV) is much higher (Creemers, 2015)[1]. We will also provide some results on the time-dependent workability uncertainty in later part of this section.

Following (Varakantham et al., 2016), we employ two evaluation metrics: 1) α-robust makespan (α-RM) output by an algorithm, and 2) Probability of Failure (PoF) which is the ratio of instances either having an actual makespan larger than α-RM (for POS) or violating any constraint (for start-time schedule). PoF is computed on a large number of $Q_s = 2000$ testing samples. Time limits for all algorithms are 10 minutes, and the returned best results are used for analysis.

5.5.1 Analysis of Our Algorithm

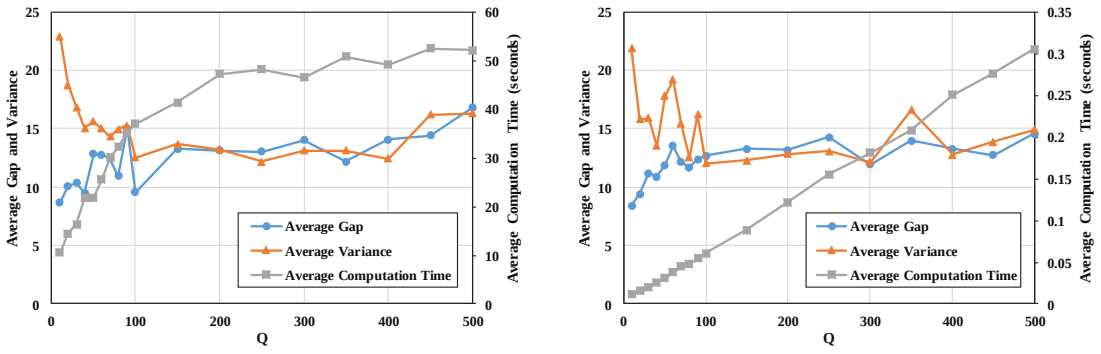
In this section, we examine our algorithms against different values of Q and α.

5.5.1.1 Impact of sample size

Since we (approximately) optimize F_β, we evaluate the impact of Q on F_β based on the gap estimator ρ of SAA, as we have done in Section 4.6.2.1. Specifically, notations $\hat{g}(\cdot)$, G_R^η and G_R^{rep} in Equation (4.19) should be replaced by $\hat{F}_\beta(\cdot)$, (G_R^η, ω^η) and $(G_R^{rep}, \omega^{rep})$, respectively. The Var_{Q_s} and $Var_{Q_{rep}}$ in Equation (4.20) should also be replaced by the variances of $\hat{F}(\cdot)$ in the Q_s times of simulations and Q_{rep} times of replications, respectively.

We plot the average gap, variance and execution time of our algorithms on a representative subset (10 instances) with Exp and $\alpha = 0.2$ in Figure 5.1. Similar to the observations in Section 4.6.2.1, we can clearly see the trade-off between solution

[1]SCV of a distribution is defined as σ^2/μ^2, where σ and μ are the standard deviation and mean, respectively.



(a) BnB-Flow (b) BnB-MCS

Figure 5.1: Results for Sample Size Test

Table 5.1: Results of BnB-Flow for Different Risk Levels

	$\mathcal{N}$				Exp			
α	α-RM	PoF	#Vio [1]	#Vio-ϵ [2]	α-RM	PoF	#Vio	#Vio-ϵ
0.2	65.26	0.13	5	0	87.55	0.19	33	5
0.15	65.97	0.08	8	0	91.83	0.15	31	2
0.1	66.15	0.06	6	0	100.33	0.09	26	1

[1] The number of instances with PoF$> \alpha$.

[2] The number of instances with PoF$> \alpha$ when $\epsilon = 0.05$.

quality and computational effort, which is expected since it is theoretically guaranteed by the properties of SAA. As shown in Figure 5.1, while the gap is relatively stable, its variance drops with the increase of Q, indicating the solution becomes more stable. The increase of execution time is not very fast, which shows good scalability of our approach on large Q. We also have similar observations in other instance subsets. In the remaining experiments, we set $Q = 100$.

Table 5.2: Results of BnB-MCS for Different Risk Levels

	$\mathcal{N}$				Exp			
α	α-RM	PoF	#Vio	#Vio-ϵ	α-RM	PoF	#Vio	#Vio-ϵ
0.2	63.4	0.13	5	0	87.33	0.19	31	8
0.15	64.44	0.09	8	0	91.54	0.15	31	3
0.1	65.49	0.06	6	0	99.95	0.09	29	0

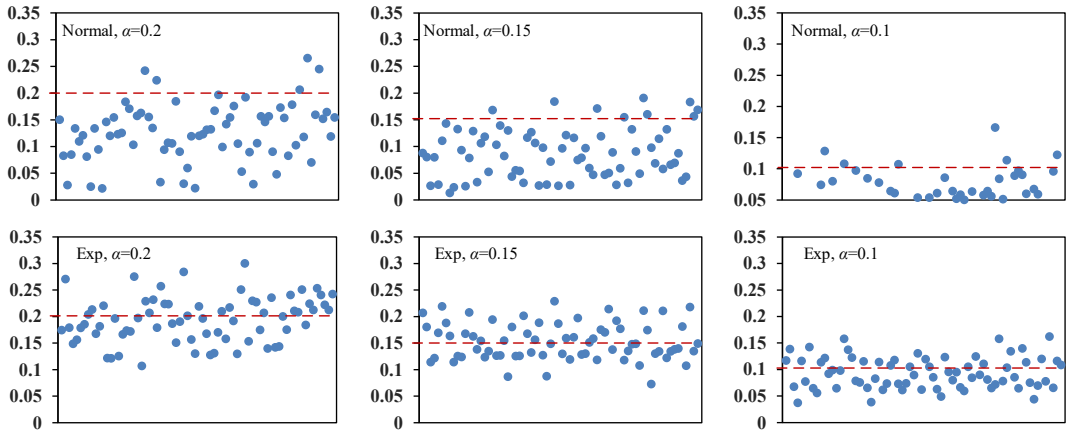


Figure 5.2: PoF Distributions of BnB-MCS for Different Risk Levels

5.5.1.2 Impact of risk parameter

To examine the impact of α, we select 72 instances by randomly picking one in each subset. In Tables 5.1 and 5.2, we present the average α-RM and PoF produced by BnB-Flow and BnB-MCS for the two distributions, with different risk levels $\alpha \in \{0.2, 0.15, 0.1\}$. As shown, the results of the two algorithms are rather close. The average computation time for BnB-Flow and BnB-MCS are 138.2 and 97.4 seconds, with 43.1% and 31.3% instances being time-out. We can observe that with stricter risk requirement (smaller α), α-RM increases since it needs to tolerate more execution scenarios. The values of α-RM is higher for Exp than $\mathcal{N}$, since the uncertainty level of the former is higher. On average, PoF is close to α which shows a precise risk control. We plot the PoF values produced by BnB-MCS in Figure 5.2, which shows most of PoFs are below the required level α.[2] But still, some instances have higher PoF than α, as shown in the columns "#Vio" of Tables 5.1 and 5.2, and Exp has more violations than $\mathcal{N}$. This is because Problem (5.5) is built on limited samples which cannot cover all situations. This is also observed in (Luedtke and Ahmed, 2008; Varakantham et al., 2016), and they propose to solve the SAA problems with stricter risk level α' than required, i.e. $\alpha' = \alpha - \epsilon$. Following this idea, we set $\epsilon = 0.05$ for our algorithms, as can be observed in the columns "#Vio-ϵ" in Tables 5.1 and 5.2 that this value can effectively reduce PoF violations. Due to the similar performance of BnB-Flow and BnB-MCS, below we will only report results of BnB-MCS for the purpose of simplicity and clarity.

[2]PoF distributions for BnB-Flow are similar as those in Figure 5.2, hence are omitted here.

Table 5.3: Number of Violations for Different ϵ Values

	$\mathcal{N}$		Exp	
ϵ	BACCHUS	SORU-H	BACCHUS	SORU-H
0	43	71	24	72
0.05	0	57	12	72
0.1	0	14	2	72

Table 5.4: Summary of Results

	$\mathcal{N}$			Exp	
	BnB-MCS	BACCHUS	SORU-H	BnB-MCS	BACCHUS
PoF$\leq \alpha$ (%)	**98.06**	98.19	86.94	**94.72**	84.44
α-RM [1]	**64.66**	65.85	68.4	**99.96**	132.71
LowestRM (%) [2]	**77.81**	26.99	7.78	**86.98**	15.1

[1] The average α-RM on instances that are successful for all algorithms.

[2] The ratio of successful instances with the lowest α-RM among all algorithms (summation may larger than 100% since different algorithms may give the same α-RM).

5.5.2 Comparison with other Approaches

Before comparing the solution quality, we first tune the parameter ϵ for BACCHUS and SORU-H. α is set to 0.2 in this section. We conduct experiments on the 72 instances used in Section 5.5.1.2, with $\epsilon \in \{0, 0.05, 0.1\}$. We report the number of violations in Table 5.3. As shown, $\epsilon = 0.05$ is reasonable for BACCHUS with a violation ratio of at most $12/72 = 16.7\%$ for both $\mathcal{N}$ and Exp. For SORU-H, $\epsilon = 0.1$ leads to satisfying results for $\mathcal{N}$, which is also the recommended value in (Varakantham et al., 2016). But for Exp which has a higher uncertainty level, all test instances are violated with $\epsilon = 0.1$. This is because SORU-H generates start-time schedule as proactive solution, which is too rigid and has a high chance to violate when the duration uncertainty level is high. In contrast, BACCHUS and our approach generate flexible solution POS, which provides better robustness (Bidot et al., 2009). In the remaining experiments, we only report the results of SORU-H on $\mathcal{N}$.

In Table 5.4, we summarize the results of the three algorithms (BnB-MCS, BAC-

CHUS and SORU-H) on all the 720 instances from Set1. We say an instance test is successful, if its PoF$\leq \alpha$. For distribution $\mathcal{N}$, BACCHUS and our algorithm BnB-MCS succeed on over 98% of the instances, which are more than SORU-H. On the 604 instances that are successful for all three algorithms, the average α-RM values are comparable. However, our algorithm achieves the lowest α-RM in over 77% of the successful instances, which is significantly higher than the other two. For distribution *Exp*, BnB-MCS succeed on nearly 95% of instances, which is higher than that of BACCHUS. On the 576 instances successfully solved by both algorithms, the average α-RM produced by our algorithm is significantly lower than that of BACCHUS, with a 25% improvement. In fact, BnB-MCS achieves lower α-RM on nearly 87% of the successful instances. We believe this performance gap is caused by the summarization heuristic used in BACCHUS. To verify our intuition, we plot the 720 PoF values produced by BACCHUS and BnB-MCS with *Exp* in Figure 5.3. As shown, PoF values of our algorithm distribute densely around the required level $\alpha = 0.2$, with 90.8% PoFs within [0.1, 0.3] and 9.2% smaller than 0.1. In contrast, PoFs of BACCHUS distribute rather sparsely, with 28% PoFs within [0.1, 0.3], 5% higher than 0.3, and nearly 67% smaller than 0.1. In addition, most of the instances with OS=0.7 have PoFs smaller than 0.1. These results indicate that the summarization heuristic tends to *over-compensate* for α, i.e. produces α-RM that is higher than required, especially for instances with higher OS. Since our algorithm solves SAA problems with tens to hundreds of samples instead of a representative one, better estimation and control of the risk level can be achieved. Similar to Chapter 4, in Table 5.5 we report the average α-RM values for *Exp* on different instance groups classified by the instance parameters. We can observe that BnB-MCS has better results on all instance groups, and the improvements on different K, RF and RC are all above 20%. However, on instances with smaller number of activities and smaller OS, the improvements are not as prominent. It would be interesting to further investigate the possible reasons of this observation to further improve the performance of our algorithm.

Finally, we briefly report the computational efficiency of the three algorithms. Note that while our algorithm BnB-MCS solves Problem (5.5) with hundreds of samples, SORU-H and BACCHUS essentially solve a much simpler deterministic RCPSP since only one summarized sample is used. In general, our algorithm BnB-MCS finds

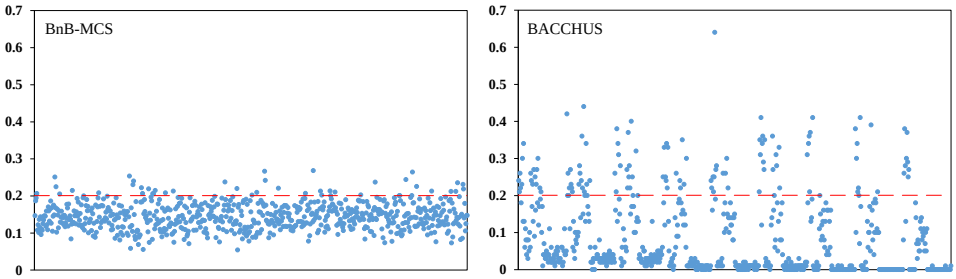


Figure 5.3: PoF Distributions for *Exp*

Table 5.5: Comparison of α-RM values for *Exp* on different instance groups

Instance group	BnB-MCS	BACCHUS	Diff(%)	Instance group	BnB-MCS	BACCHUS	Diff(%)
N=10	**61.48**	75.2	18.24	OS=0.2	**80.29**	93.03	13.7
N=20	**98.42**	129.92	24.25	OS=0.7	**113.71**	159.76	28.82
N=30	**134.64**	183.87	26.78	RF=0.2	**87.71**	114.48	23.38
K=1	**94.67**	126.09	24.92	RF=0.7	**112.72**	150.89	25.29
K=2	**100.67**	132.01	23.74	RC=0.2	**93.53**	125.23	25.31
K=3	**104.88**	139.26	24.69	RC=0.7	**104.85**	137.69	23.85

the optimal solutions of Problem (5.5) for over 70% of all instances with an average computation time of 93.02 seconds, while SORU-H and BACCHUS optimally solve nearly 90% of the corresponding deterministic instances in 8.38 seconds on average. However, it is common in the experiments that a sub-optimal solution given by BnB-MCS is much better than the optimal solutions given by SORU-H and BACCHUS. We plot the percentage of time-out instances with the problem parameters in Figure 5.4. As shown, the three algorithms share the same trend, i.e. the hardness increases with N, K, RF and RC, but decreases with RF. This corresponds to our observation in Section 4.6.2.3, and can be explained by similar reasons.

5.5.3 Results on Time-dependent Workability Uncertainty

In this section, we conduct experiments on instances with time-dependent workability uncertainty modeled using the dataset in Table 4.1. We run our algorithm BnB-MCS on the 72 instances used in Section 5.5.1.2 with elongated durations, and set the risk level $\alpha = 0.2$. In Table 5.6, we summary the results according to the

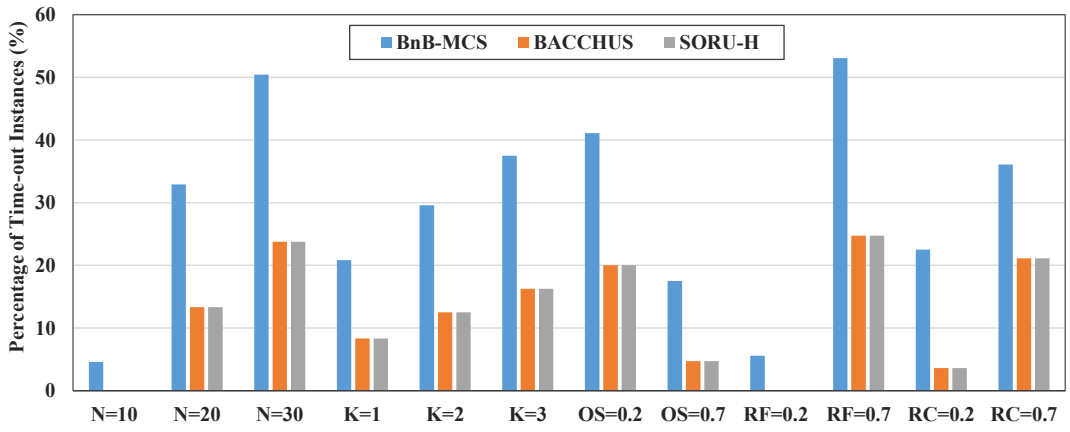


Figure 5.4: Percentage of Time-out Instances

Table 5.6: Comparison of Expected and Robust Makespan for Time-dependent Workability Uncertainty

Instance group	Expected Makespan	Robust Makespan	Instance group	Expected Makespan	Robust Makespan
N=10	582.28	600.29	OS=0.2	788.95	801.17
N=20	583.99	598.21	OS=0.7	473.02	503.25
N=30	726.68	758.13	RF=0.2	511.77	540.36
K=1	572.51	601.58	RF=0.7	750.2	764.06
K=2	630.14	652.13	RC=0.2	559.07	576.61
K=3	690.3	702.92	RC=0.7	702.9	727.81

instance parameters, along with the corresponding expected makespan values. As shown in this table, the robust makespan are larger than the expected makespan, which is reasonable since the probability of violation needs to be controlled as specified by α. We also plot the PoF values for the 72 instances in Figure 5.5. As shown, our algorithm successfully controls the risk level for most of the instances (70 out of 72) as required by α. Therefore, our approach can also effectively solve the risk-aware proactive scheduling problems with the time-dependent workability uncertainty, which cannot be solved by existing risk-aware approaches.

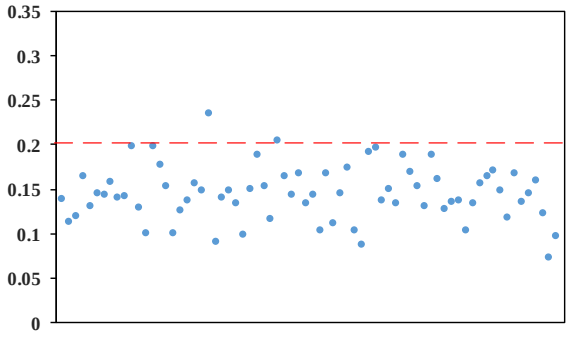


Figure 5.5: PoF Distribution for Time-dependent Workability Uncertainty

5.6 Conclusions

In this chapter, we propose a novel approach for risk-aware project scheduling, by exploiting a mathematically coherent risk measure CVaR. Since previous CVaR optimization approach cannot be applied to our proactive problem due to the combinatorial nature of RCPSP, we design a general branch-and-bound framework with efficient bound computation for combinatorial CVaR minimization. We then instantiate it to solve the proactive scheduling problem with the components we designed for the risk-neutral problem in the previous chapter. Empirical results show that our approach scales well to a large number of samples, and provides better risk control and robust makespan than state-of-the-art approaches.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

In this thesis, we have provided effective approaches to address complex RCPSP with two important practical factors, namely *distributed management* and *execution uncertainty*. Under these factors, traditional RCPSP approaches with the assumptions of centralization and static execution environment are invalidated. However, incorporating these practical factors requires carefully designed algorithms to achieve good performance, since RCPSP is already computationally intractable. Our contributions in designing novel algorithms for addressing these challenging problems are summarized below.

In Chapter 3, we address the problem of distributed multi-project scheduling, formally named as DRCMPSP. Different from RCPSP, this problem involves multiple decision makers (project agents) in the scheduling process, which requires an algorithm that can schedule these projects in a distributed manner, and respect the information privacy of the project agents. We consider the multi-project scheduling problem as a multi-agent resource allocation problem, and formulate it as a multi-unit combinatorial auction named as DRCMPSP auction. In such an auction process, the project agents bid for the shared resources controlled by a mediator agent. Some existing works have applied combinatorial auction on similar problems as DRCMPSP, but with much simpler settings (e.g. single-unit resource, small activity sets). In addition, formal analysis on the relations between scheduling and auction is rather rare. We first fill this gap by theoretically proving that under

reasonable assumption (i.e. revenue of each project agent is sufficiently high), an optimal multi-project schedule can be found by finding an optimal allocation of the auction. However, to participate the DRCMPSP auction, the bidders need to evaluate different resource allocations, which are intractable problems due to the combinatorial nature of RCPSP. Therefore, we propose to elicit useful valuation information from the bidders by using *capacity query*, which can be answered by solving an RCPSP with time-varying resource capacities. Based on the capacity query, we design two allocation strategies for efficiently finding good resource allocations, including a greedy allocation strategy and a branch-and-bound heuristic. Empirical results show that both strategies can produce high quality solutions that are better than state-of-the-art approaches, and scale well to large problem instances with tens of projects and thousands of activities.

In Chapter 4, we study the risk-neutral proactive scheduling problem of RCPSP with uncertain activity durations. More specifically, we aim at finding an (approximately) optimal POS with the minimal expected makespan. Most of the existing proactive scheduling approaches assume that the uncertain duration of an activity can be modeled as a time-independent random variable. We relax this limited assumption and allow the existence of the *time-dependent workablity uncertainty*, which causes the random activity durations to be time-dependent. To the best of our knowledge, this type of uncertainty has never been studied in the literature. In this work, we first generalize the traditional time-independent duration model to support the time-dependent workability uncertainty. We then formulate the proactive scheduling problem as a stochastic optimization problem on this generalized model, which naturally generalizes the traditional time-independent proactive problem. To solve the challenging stochastic optimization problem, we propose to approximate it based on SAA which provides asymptotical guarantee of converging to the optimal solution. To optimally solve the NP-hard SAA problem, we design two branch-and-bound algorithms that search for the optimal POS from different angles, by exploiting interesting problem properties. We conduct rich experiments on multiple uncertain duration models, including pure time-dependent, pure time-independent, and mixture models with both components. Results confirm that our algorithms can find high-quality proactive solutions on a variety of uncertainty distributions.

In Chapter 5, we focus on the proactive scheduling problem with a risk-aware

objective, i.e. minimization of the robust makespan. This objective is more practical in risk-aware settings than minimization of the expected makespan, since it controls the probability that the actual makespan exceeds certain value within a predefined risk level, while the expected makespan may be violated with a high chance during execution. From the perspective of stochastic optimization, this objective corresponds to the minimization of VaR, which is commonly solved by applying SAA on a chance-constrained optimization problem. However, directly optimizing VaR for the proactive scheduling problem leads to intractable MILPs that scales poorly with the sample size. Alternatively, we propose to solve the proactive problem by optimizing CVaR, which is mathematically coherent and more tractable than optimizing VaR. Due to the involvement of RCPSP, the solution space of our problem is combinatorial, which excludes the application of traditional CVaR optimization approaches since they normally assume linear solution spaces. We then propose a general branch-and-bound framework for combinatorial CVaR optimization, which is equipped with a computationally efficient lower bound and requires only a sample bounding function. We adapt the two algorithms designed in Section 4 with this general framework, and perform numerical studies to verify the effectiveness of our algorithms. Experimental results show that our algorithms scale well to large numbers of samples, and can provide better solution quality and risk management than state-of-the-art approaches.

Finally, we give a brief discussion on the applicability of different techniques proposed in this thesis. When the scheduling problem is deterministic and contains multiple projects that are controlled by multiple agents in a distributed manner, the greedy and branch-and-bound strategies in Chapter 3 can be applied. Given the good scalability, large problem instances with tens of projects and thousands of activities can be solved within reasonable computational time. When the scheduling problem is centralized with random activity durations, the algorithms in Chapters 4 and 5 are useful. Depending on the requirement, the decision maker can choose either of the risk-neutral and risk-aware versions of the algorithms. Small and median-sized problem instances with tens of activities can be solved, but large problem instances may take very long time to solve, since the algorithms are (approximately) complete with the proven ability of finding the (approximately) optimal solutions.

6.2 Future Work

In this section, we identify several promising directions for extending our current research as future work.

6.2.1 Distributed Resource Allocation and Scheduling under Uncertainty

The DRCMPSP model studied in Chapter 3 is deterministic. It would be very interesting to investigate the distributed scheduling problem under uncertainty, which considers both distributed management and execution uncertainty at the same time. However, this comes with several challenges. Firstly, as we have demonstrated in Chapter 3, a large number of activities may be involved in a multi-project setting, which makes it impractical to find the optimal solutions. Instead, we should focus on developing approximate algorithms that can produce high quality solutions within reasonable computation time. Secondly, we need to find a suitable way to represent the solutions. For single project and centralized multi-project problems, POS is an ideal flexible solution. However, in a distributed setting, the representation and generation of POS should be conducted in a distributed manner, since no agent have full control of all the activities. The third issue is about how to represent the preferences of the agents. Depending on the solution representation, preference of project agents may be defined and evaluated on different graphs, instead of on the multisets of items. These three issues should be addressed together in designing efficient distributed algorithms. Intuitively, theory and methods in the Distributed Constraint Optimization Problems (DCOP) (Rogers et al., 2011; Ghosh et al., 2015) may be suitable foundations for building practical solutions to this problem.

6.2.2 Designing Stronger Sample Bounding Functions

In Section 4 and 5 which study proactive scheduling techniques, we have designed several branch-and-bound algorithms for solving the corresponding SAA problems. Though they can solve most of the test instances optimally, there is potential in further improving the computational efficiency to reduce the average time and close more instances. One promising direction is to design stronger sample bounding functions, which are common components that are used in both the risk-neutral and

risk-aware settings. Our current sample bounding functions only use the original precedence constraints to compute the lower bounds. It would be interesting to design stronger lower bounding functions that explicitly incorporate the resource constraints. A possible way is to conduct temporal constraint propagation on each sample, which requires extending the traditional constraint propagation techniques, such as edge-finding and energetic reasoning (Laborie, 2003), to our problem setting.

6.2.3 Incorporating General Temporal Relations

Currently, the temporal constraints in our problem models are all precedence constraints. Though they are useful in modeling the temporal relations between activities, in general, more types of temporal constraints may exist. More technically, precedence constraints are commonly referred to as Finish-to-Start relations (Neumann et al., 2012), since they require an activity to start after the completion of another one. Similarly, Start-to-Start, Start-to-Finish, and Finish-to-Finish relations can also be defined for two activities. Though these temporal relations can be easily converted from one to another (Varakantham et al., 2016), our current methods and algorithms are not directly applicable to problems with general temporal relations. Nevertheless, it is worth extending our approaches to support these general temporal constraints, since they exist widely in real-world problems and can greatly improve the usefulness and generality of our algorithms.

Bibliography

Adhau, S., Mittal, M., 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.

Amir, O., Sharon, G., and Stern, R. (2015). Multi-agent pathfinding as a combinatorial auction. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI'15)*, pages 2003–2009.

Artigues, C., Michelon, P., and Reusser, S. (2003). Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, 149(2):249–267.

Ashtiani, B., Leus, R., and Aryanezhad, M.-B. (2011). New competitive results for the stochastic resource-constrained project scheduling problem: exploring the benefits of pre-processing. *Journal of Scheduling*, 14(2):157–171.

Ausubel, L. M. and Milgrom, P. R. (2002). Ascending auctions with package bidding. *Frontiers of Theoretical Economics*, 1(1).

Baarslag, T. and Gerding, E. H. (2015). Optimal incremental preference elicitation during negotiation. In *Proceedings of the Twenty-Fourth International Conference on Artificial Intelligence (IJCAI'15)*, pages 3–9. AAAI Press.

Ballestín, F. (2007). When it is worthwhile to work with the stochastic rcpsp? *Journal of Scheduling*, 10(3):153–166.

Baptiste, P., Le Pape, C., and Nuijten, W. (2012). *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer Science & Business Media.

Beck, J. C. and Wilson, N. (2007). Proactive algorithms for job shop scheduling with probabilistic durations. *Journal of Artificial Intelligence Research*, 28:183–232.

Beşikci, U., Bilge, Ü., and Ulusoy, G. (2015). Multi-mode resource constrained multi-project scheduling and resource portfolio problem. *European Journal of Operational Research*, 240(1):22–31.

Bidot, J., Vidal, T., Laborie, P., and Beck, J. C. (2009). A theoretic and practical framework for scheduling in a stochastic environment. *Journal of Scheduling*, 12(3):315–344.

Blazewicz, J., Lenstra, J. K., and Kan, A. R. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24.

Blumrosen, L. and Nisan, N. (2009). On the computational power of demand queries. *SIAM Journal on Computing*, 39(4):1372–1391.

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.

Brucker, P. and Knust, S. (2012). *Complex Scheduling*. Springer.

Bruni, M. E., Beraldi, P., and Guerriero, F. (2015). The stochastic resource-constrained project scheduling problem. In *Handbook on Project Management and Scheduling Vol. 2*, pages 811–835. Springer.

Clarke, E. H. (1971). Multipart pricing of public goods. *Public choice*, 11(1):17–33.

Conen, W. and Sandholm, T. (2001). Preference elicitation in combinatorial auctions. In *Proceedings of the 3rd ACM conference on Electronic Commerce (EC'01)*, pages 256–259. ACM.

Confessore, G., Giordani, S., and Rismondo, S. (2007). A market-based multi-agent system model for decentralized multi-project scheduling. *Annals of Operations Research*, 150(1):115–135.

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

Cui, J., Yu, P., Fang, C., Haslum, P., and Williams, B. C. (2015). Optimising bounds in simple temporal networks with uncertainty under dynamic controllability constraints. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS'15)*, pages 52–60.

Davenport, A., Gefflot, C., and Beck, C. (2001). Slack-based techniques for robust schedules. In *Sixth European Conference on Planning (ECP)*, pages 43–49.

de Nijs, F. and Klos, T. (2014). A novel priority rule heuristic: Learning from justification. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS'14)*, pages 92–100.

Demange, G., Gale, D., and Sotomayor, M. (1986). Multi-item auctions. *The Journal of Political Economy*, pages 863–872.

Demeulemeester, E. L. and Herroelen, W. S. (1997). New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43(11):1485–1492.

Fink, A. and Homberger, J. (2015). Decentralized multi-project scheduling. In *Handbook on Project Management and Scheduling Vol. 2*, pages 685–706. Springer.

Fu, N., Lau, H. C., and Varakantham, P. (2015). Robust execution strategies for project scheduling with unreliable resources and stochastic durations. *Journal of Scheduling*, 18(6):607–622.

Fu, N., Lau, H. C., Varakantham, P., and Xiao, F. (2012). Robust local search for solving rcpsp/max with durational uncertainty. *Journal of Artificial Intelligence Research*, 43:43–86.

Fu, N., Varakantham, P., and Lau, H. C. (2016). Robust partial order schedules for rcpsp/max with durational uncertainty. In *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling (ICAPS'16)*, pages 124–130.

Ghosh, S., Kumar, A., and Varakantham, P. (2015). Probabilistic inference based message-passing for resource constrained dcops. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 411–417.

Gonen, R. and Lehmann, D. (2000). Optimal solutions for multi-unit combinatorial auctions: Branch and bound heuristics. In *Proceedings of the 2nd ACM conference on Electronic Commerce (EC'00)*, pages 13–20.

Groves, T. (1973). Incentives in teams. *Econometrica: Journal of the Econometric Society*, pages 617–631.

Gul, F. and Stacchetti, E. (2000). The english auction with differentiated commodities. *Journal of Economic theory*, 92(1):66–95.

Gurobi Optimization, I. (2015). Gurobi optimizer reference manual.

Hagstrom, J. N. (1988). Computational complexity of pert problems. *Networks*, 18(2):139–147.

Hartmann, S. and Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of operational research*, 207(1):1–14.

Herroelen, W. and Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials. *European journal of operational research*, 165(2):289–306.

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

Hong, L. J., Hu, Z., and Liu, G. (2014). Monte carlo methods for value-at-risk and conditional value-at-risk: A review. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 24(4):22.

Hudson, B. and Sandholm, T. (2004). Effectiveness of query types and policies for preference elicitation in combinatorial auctions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1 (AAMAS'04)*, pages 386–393.

Igelmund, G. and Radermacher, F. J. (1983). Preselective strategies for the optimization of stochastic project networks under resource constraints. *Networks*, 13(1):1–28.

Kang, D., Bing, Z. C., Song, W., Hu, Z., Chen, S., Zhang, J., and Xi, H. (2017). Automatic construction of agent-based simulation using business process diagrams and ontology-based models (demo). In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS'17)*, pages 1793–1795.

Kelley Jr, J. E. (1961). Critical-path planning and scheduling: Mathematical basis. *Operations research*, 9(3):296–320.

Kelso Jr, A. S. and Crawford, V. P. (1982). Job matching, coalition formation, and gross substitutes. *Econometrica: Journal of the Econometric Society*, pages 1483–1504.

Kleywegt, A. J., Shapiro, A., and Homem-de Mello, T. (2002). The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502.

Kolisch, R. (1996). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2):320–333.

Kolisch, R. and Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European journal of operational research*, 174(1):23–37.

Koné, O., Artigues, C., Lopez, P., and Mongeau, M. (2011). Event-based milp models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3–13.

Krysta, P., Telelis, O., and Ventre, C. (2013). Mechanisms for multi-unit combinatorial auctions with a few distinct goods. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'13)*, pages 691–698.

Kurtulus, I. and Davis, E. (1982). Multi-project scheduling: Categorization of heuristic rules performance. *Management Science*, 28(2):161–172.

Kutanoglu, E. and Wu, S. D. (1999). On combinatorial auction and lagrangean relaxation for distributed resource scheduling. *IIE transactions*, 31(9):813–826.

Laborie, P. (2003). Algorithms for propagating resource constraints in ai planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 143(2):151–188.

Laborie, P. (2005). Complete mcs-based search: Application to resource constrained project scheduling. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 181–186.

Lamas, P. and Demeulemeester, E. (2016). A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations. *Journal of Scheduling*, 19(4):409–428.

Lambrechts, O., Demeulemeester, E., and Herroelen, W. (2011). Time slack-based techniques for robust project scheduling subject to resource uncertainty. *Annals of Operations Research*, 186(1):443–464.

Larson, K. and Sandholm, T. (2001). Costly valuation computation in auctions. In *Proceedings of the 8th conference on Theoretical aspects of rationality and knowledge*, pages 169–182. Morgan Kaufmann Publishers Inc.

Lau, J. S., Huang, G. Q., Mak, K.-L., and Liang, L. (2006). Agent-based modeling of supply chains for distributed scheduling. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 36(5):847–861.

Leus, R. and Herroelen, W. (2004). Stability and resource allocation in project planning. *IIE transactions*, 36(7):667–682.

Liao, T. W., Egbelu, P., Sarker, B., and Leu, S. (2011). Metaheuristics for project and construction management–a state-of-the-art review. *Automation in construction*, 20(5):491–505.

Lombardi, M. and Milano, M. (2012). A min-flow algorithm for minimal critical set detection in resource constrained project scheduling. *Artificial Intelligence*, 182:58–67.

Lombardi, M., Milano, M., and Benini, L. (2013). Robust scheduling of task graphs under execution time uncertainty. *IEEE transactions on computers*, 62(1):98–111.

Lova, A. and Tormos, P. (2001). Analysis of scheduling schemes and heuristic rules performance in resource-constrained multiproject scheduling. *Annals of Operations Research*, 102(1-4):263–286.

Luedtke, J. and Ahmed, S. (2008). A sample approximation approach for optimization with probabilistic constraints. *SIAM Journal on Optimization*, 19(2):674–699.

Mao, X., Roos, N., and Salden, A. (2009). Stable multi-project scheduling of airport ground handling services by heterogeneous agents. In *Proceedings of The 8th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'09)-Volume 1*, pages 537–544.

Möhring, R. H. (2000). Scheduling under uncertainty: Optimizing against a randomizing adversary. *Edited by G. Goos, J. Hartmanis and J. van Leeuwen*, page 15.

Morris, P. and Muscettola, N. (2005). Temporal dynamic controllability revisited. In *Proceedings of the Twentieth national conference on Artificial intelligence (AAAI'05)*, pages 1193–1198.

Morris, P., Muscettola, N., and Vidal, T. (2001). Dynamic control of plans with temporal uncertainty. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 494–499.

Naber, A. and Kolisch, R. (2014). Mip models for resource-constrained project scheduling with flexible resource profiles. *European Journal of Operational Research*, 239(2):335–348.

Neumann, K., Schwindt, C., and Zimmermann, J. (2012). *Project scheduling with time windows and scarce resources: temporal and resource-constrained project scheduling with regular and nonregular objective functions*. Springer Science & Business Media.

Nisan, N. and Ronen, A. (2007). Computationally feasible vcg mechanisms. *Journal of Artificial Intelligence Research*, 29:19–47.

Parkes, D. C. (2005). Auction design with costly preference elicitation. *Annals of Mathematics and Artificial Intelligence*, 44(3):269–302.

Parkes, D. C. and Ungar, L. H. (2000). Iterative combinatorial auctions: Theory and practice. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'00)*, pages 74–81.

Parkes, D. C. and Ungar, L. H. (2001). An auction-based method for decentralized train scheduling. In *Proceedings of the 5th International Conference on Autonomous Agents (AAMAS'01)*, pages 43–50.

Policella, N., Cesta, A., Oddi, A., and Smith, S. F. (2009). Solve-and-robustify. *Journal of Scheduling*, 12(3):299–314.

Policella, N., Smith, S. F., Cesta, A., and Oddi, A. (2004). Generating robust schedules through temporal flexibility. In *Proceedings of the Fourteenth International Conference on International Conference on Automated Planning and Scheduling (ICAPS'04)*, pages 209–218.

Pritsker, A. A. B., Waiters, L. J., and Wolfe, P. M. (1969). Multiproject scheduling with limited resources: A zero-one programming approach. *Management science*, 16(1):93–108.

Rockafellar, R. T. and Uryasev, S. (2002). Conditional value-at-risk for general loss distributions. *Journal of banking & finance*, 26(7):1443–1471.

Rogers, A., Farinelli, A., Stranders, R., and Jennings, N. R. (2011). Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, 175(2):730–759.

Rossi, F., Van Beek, P., and Walsh, T. (2006). *Handbook of constraint programming*. Elsevier.

Sandholm, T. (2002). Algorithm for optimal winner determination in combinatorial auctions. *Artificial intelligence*, 135(1):1–54.

Schutt, A., Feydy, T., Stuckey, P. J., and Wallace, M. G. (2013). Solving rcpsp/max by lazy clause generation. *Journal of Scheduling*, 16(3):273–289.

Schwindt, C., Zimmermann, J., et al. (2015). Handbook on project management and scheduling vol. 1. *Cham: Springer International Publishing*.

Song, W., Kang, D., Zhang, J., and Xi, H. (2016). Decentralized multi-project scheduling via multi-unit combinatorial auction. In *Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems (AAMAS'16)*, pages 836–844.

Song, W., Kang, D., Zhang, J., and Xi, H. (2017a). A multi-unit combinatorial auction based approach for decentralized multi-project scheduling. *Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 31(6):1548–1577.

Song, W., Kang, D., Zhang, J., and Xi, H. (2017b). Proactive project scheduling with time-dependent workability uncertainty. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS'17)*, pages 221–229.

Song, W., Kang, D., Zhang, J., and Xi, H. (2017c). A sampling based approach for proactive project scheduling with time-dependent duration uncertainty (student abstract). In *Proceedings of the Thirty-first AAAI Conference on Artificial Intelligence (AAAI'17)*, pages 4985–4986.

Song, W., Kang, D., Zhang, J., and Xi, H. (2018). Risk-aware proactive scheduling via conditional value-at-risk. In *Proceedings of the Thirty-second AAAI Conference on Artificial Intelligence (AAAI'18, to appear)*.

Stork, F. (2001). Stochastic resource-constrained project scheduling.

Stranjak, A., Dutta, P. S., Ebden, M., Rogers, A., and Vytelingum, P. (2008). A multi-agent simulation system for prediction and scheduling of aero engine overhaul. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'08): Industrial Track*, pages 81–88.

Van Peteghem, V. and Vanhoucke, M. (2014). An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling

problem on new dataset instances. *European Journal of Operational Research*, 235(1):62–72.

Vanhoucke, M., Coelho, J., Debels, D., Maenhout, B., and Tavares, L. V. (2008). An evaluation of the adequacy of project network generators with systematically sampled networks. *European Journal of Operational Research*, 187(2):511–524.

Varakantham, P., Fu, N., and Lau, H. C. (2016). A proactive sampling approach to project scheduling under uncertainty. In *Proceedings of the Thirtieth national conference on Artificial intelligence (AAAI'16)*, pages 3195–3201.

Vickrey, W. (1961). Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance*, 16(1):8–37.

Vytelingum, P., Rogers, A., Macbeth, D. K., Dutta, P., Stranjak, A., and Jennings, N. R. (2009). A market-based approach to multi-factory scheduling. In *International Conference on Auctions, Market Mechanisms and Their Applications*, pages 74–86. Springer.

Walsh, W. E. and Wellman, M. P. (2003). Decentralized supply chain formation: A market protocol and competitive equilibrium analysis. *Journal of Artificial Intelligence Research*, 19:513–567.

Wellman, M. P., Walsh, W. E., Wurman, P. R., and MacKie-Mason, J. K. (2001). Auction protocols for decentralized scheduling. *Games and economic behavior*, 35(1):271–303.

Xi, H., Goh, C. K., Dutta, P. S., Sha, M., and Zhang, J. (2015). An agent-based simulation system for dynamic project scheduling and online disruption resolving. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (AAMAS'15)*, pages 1759–1760.

Zheng, Z., Guo, Z., Zhu, Y., and Zhang, X. (2014). A critical chains based distributed multi-project scheduling approach. *Neurocomputing*, 143:282–293.

Publications

Kang, D., Bing, Z. C., Song, W., Hu, Z., Chen, S., Zhang, J., and Xi, H. (2017). Automatic construction of agent-based simulation using business process diagrams and ontology-based models (demo). In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS'17)*, pages 1793–1795.

Song, W. (2016). An auction-based approach for decentralized multi-project scheduling (doctoral consortium). In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems (AAMAS'16)*, pages 1518–1519.

Song, W. (2017). Project scheduling in complex business environments (doctoral consortium). In *Proceedings of the Thirty-first AAAI Conference on Artificial Intelligence (AAAI'17)*, pages 5052–5053.

Song, W., Kang, D., Zhang, J., and Xi, H. (2016). Decentralized multi-project scheduling via multi-unit combinatorial auction. In *Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems (AAMAS'16)*, pages 836–844.

Song, W., Kang, D., Zhang, J., and Xi, H. (2017a). A multi-unit combinatorial auction based approach for decentralized multi-project scheduling. *Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 31(6):1548–1577.

Song, W., Kang, D., Zhang, J., and Xi, H. (2017b). Proactive project scheduling with time-dependent workability uncertainty. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS'17)*, pages 221–229.

Song, W., Kang, D., Zhang, J., and Xi, H. (2017c). A sampling based approach for proactive project scheduling with time-dependent duration uncertainty (stu-

dent abstract). In *Proceedings of the Thirty-first AAAI Conference on Artificial Intelligence (AAAI'17)*, pages 4985–4986.

Song, W., Kang, D., Zhang, J., and Xi, H. (2018). Risk-aware proactive scheduling via conditional value-at-risk. In *Proceedings of the Thirty-second AAAI Conference on Artificial Intelligence (AAAI'18)*, pages 6278–6285.

Appendix A

Fast Bid Generation Algorithm

Here we design a fast polynomial-time algorithm to solve the RCPSP with time-varying resource capacities, as shown in Algorithm 7. This algorithm is based on the parallel generation scheme (Kolisch, 1996). The original algorithm in (Kolisch, 1996) is designed for resources with constant capacity over the whole scheduling horizon. Here we modify it to incorporate the time-varying resource capacities.

Intuitively, Algorithm 7 consists of a series of stages at certain time tic t_d. Three sets of activities are maintained during the whole process: Active Set $\overline{AS}$, Complete Set $\overline{CS}$, and Decision Set $\overline{DS}$. In each stage, firstly the activities in $\overline{AS}$ that complete before t_d are moved from $\overline{AS}$ to $\overline{CS}$. Next, a set of activities that can be scheduled with respect to precedence and resource constraints are identified and put into $\overline{DS}$. Then, an iterative process is imposed on $\overline{DS}$, which includes three steps: 1) choose the activity a_{ij} according to certain priority rule to start at t_d, 2) move a_{ij} from $\overline{DS}$ to $\overline{AS}$, and 3) update the activities in $\overline{DS}$. This process terminates when no more activity can be scheduled in t_d, i.e. $\overline{DS} = \emptyset$, which leads to the update of t_d. If $\overline{AS}$ is not empty, then the new t_d is set to be the minimum complete time of the activities in $\overline{AS}$. In the case $\overline{AS}$ is empty, which indicates no remaining activity can start at t_d (due to insufficient resource), t_d is set to the next time tic. When all activities are scheduled, the algorithm terminates with a feasible schedule.

The operation of updating $\overline{DS}$ in Line 4 and 8 includes two steps: 1) find the unscheduled activities whose predecessors are all in $\overline{CS}$, and 2) exclude those activities a_{ij} that cannot be scheduled to start at t_d due to insufficient resource at some time slot $t_d \leq t \leq t_d + d_{ij}$. Different priority rules (e.g. Latest Finish Time (LFT), Most Total Successors (MTS), Minimum Slack (MS)) can be used in Line 6 to select an

activity. Among these priority rules, LFT has been empirically shown to be the most effective one in minimizing the project delay Kolisch (1996) and is chosen for our scheduling algorithm. Complexity of Algorithm 7 is $O\left(N_i^2(K+L_i)d_i^*\right)$ (ignore the update of t_d between Line 9 and 11), where $d_i^* = \max\{d_{i1}, ..., d_{iJ_i}\}$ is the maximum duration of the activities of P_i.

Algorithm 7: Fast algorithm for solving RCPSP with time-varying resource capacities

Input: A project P_i, scheduling horizon T, local resource capacity profile $[C_{l_i t}]_{L_i \times T}$, global resource capacity profile $[\Psi_{kt}]_{K \times T}$

Output: A feasible schedule $S_i = \{s_{i1}, ..., s_{iN_i}\}$

1 Initialization: $t_d \leftarrow ed_i$, $\overline{AS} \leftarrow \emptyset$, $\overline{CS} \leftarrow \emptyset$, $\overline{DS} \leftarrow \emptyset$;
2 **while** $|\overline{AS}| + |\overline{CS}| < J_i$ **do**
3 $\overline{CS} \leftarrow \overline{CS} \bigcup \{a_{ij} \in \overline{AS} | s_{ij} + d_{ij} \leq t_d\}$, $\overline{AS} \leftarrow \overline{AS} \backslash \{a_{ij} \in \overline{AS} | s_{ij} + d_{ij} \leq t_d\}$;
4 Update $\overline{DS}$;
5 **while** $\overline{DS} \neq \emptyset$ **do**
6 Select one activity $a_{ij} \in \overline{DS}$ according to a priority rule, and $s_{ij} \leftarrow t_d$;
7 $\overline{DS} \leftarrow \overline{DS} \backslash \{a_{ij}\}$, $\overline{AS} \leftarrow \overline{AS} \bigcup \{a_{ij}\}$;
8 Update $\overline{DS}$;
9 **if** $\overline{AS} = \emptyset$ **then**
10 $t_d \leftarrow t_d + 1$;
11 **else**
12 $t_d \leftarrow \min\{s_{ij} + d_{ij} | a_{ij} \in \overline{AS}\}$;
13 **return** S_i