Journal of Artificial Intelligence Research 64 (2019) 385-427 Submitted 04/18; published 02/19

A Sampling Approach for Proactive Project Scheduling under Generalized Time-dependent Workability Uncertainty

Wen Song SONGWEN@NTU.EDU.SG
Donghun Kang DONGHUN.KANG@NTU.EDU.SG
Rolls-Royce@NTU Corp Lab
Nanyang Technological University
50 Nanyang Avenue, Singapore 639798

Jie Zhang ZHANGJ@NTU.EDU.SG
School of Computer Science and Engineering
Nanyang Technological University
50 Nanyang Avenue, Singapore 639798

Zhiguang Cao ISECAOZ@NUS.EDU.SG
Department of Industrial Systems Engineering and Management
Centre for Maritime Studies
National University of Singapore
1 Engineering Drive 2, Singapore 119077

Hui Xi HUI.XI@ROLLS-ROYCE.COM
Rolls-Royce Singapore Pte Ltd
6 Seletar Aerospace Rise, Singapore 797575

Abstract

In real-world project scheduling applications, activity durations are often uncertain. Proactive scheduling can effectively cope with the duration uncertainties, by generating robust baseline solutions according to a priori stochastic knowledge. However, most of the existing proactive approaches assume that the duration uncertainty of an activity is not related to its scheduled start time, which may not hold in many real-world scenarios. In this paper, we relax this assumption by allowing the duration uncertainty to be time-dependent, which is caused by the uncertainty of whether the activity can be executed on each time slot. We propose a stochastic optimization model to find an optimal Partial-order Schedule (POS) that minimizes the expected makespan. This model can cover both the time-dependent uncertainty studied in this paper and the traditional time-independent duration uncertainty. To circumvent the underlying complexity in evaluating a given solution, we approximate the stochastic optimization model based on Sample Average Approximation (SAA). Finally, we design two efficient branch-and-bound algorithms to solve the NP-hard SAA problem. Empirical evaluation confirms that our approach can generate high-quality proactive solutions for a variety of uncertainty distributions.

1. Introduction

Resource-Constrained Project Scheduling Problem (RCPSP) is a general model for various scheduling applications in manufacturing, logistics and business management. An instance of RCPSP contains a set of non-preemptive activities and multiple renewable resources with limited capacities. Each activity has certain amounts of requirement for one or more types of resources, and may have precedence dependencies with other activities. Traditional research on RCPSP assumes a deterministic problem setting where the activity durations are fully known as static values before execution (Demeulemeester & Herroelen, 1997; Laborie, 2005; Song, Kang, Zhang, & Xi, 2017a). However, real-world activities are highly sensitive to various sources of uncertainties (e.g. transportation time, manpower availability, weather condition changes). Consequently, the activity durations become uncertain, which may further lead to frequent disruptions of schedules generated by solving deterministic RCPSP instances. Therefore, it is of great practical value to account for the uncertainties in solving the scheduling problems.

In the literature, many approaches have considered the problem of scheduling in stochastic environment. According to the taxonomy proposed by Bidot, Vidal, Laborie, and Beck (2009), these approaches can be classified into three groups, namely *proactive*, *revision*, and *progressive* approaches. More specifically, proactive approach refers to those techniques that exploit a prior stochastic knowledge to generate a *baseline* solution before execution, which can be a complete schedule or a policy that will be used during execution to generate a schedule. The baseline solution will never be modified in execution. In contrast, revision techniques generate a complete schedule before execution, and revise it whenever needed according to actual observations (e.g. schedule becomes infeasible, quality degrades too much). Techniques in the last group, i.e. progressive approaches, do not generate any solution before execution, instead all the scheduling decisions are made in an online fashion during execution. Compared with the other two alternatives, proactive approaches tend to produce solutions with higher quality and robustness, since the stochastic knowledge is explicitly taken into account in generating the baseline solutions (Bidot et al., 2009). Moreover, because the decisions made in the baseline solution (e.g. activity start times, activity dependencies, resource allocation commitments) will not be revised online, it can serve as an important guideline in coordinating various aspects of the whole execution process, for example material preparation, supply chain planning, subtask planning, and so on (Lamas & Demeulemeester, 2016; Aytug, Lawley, McKay, Mohan, & Uzsoy, 2005). Thus, we study the proactive scheduling problem of RCPSP in this paper.

Till now, a number of proactive scheduling 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 the impact of all the uncertainty sources (Bruni, Beraldi, & Guerriero, 2015). As a motivating example, suppose an activity can be executed (i.e. workable) only when certain weather conditions (e.g. temperature, humidity, wind speed, and so on) are satisfied; meanwhile, the activity needs to secure enough workable days to finish successfully. In this case, the activity duration uncertainty comes from the stochastic *workability* of each time slot (day). Due to the seasonality of weather conditions, the duration uncertainty of an activity should depend on its start time, which contradicts the assumption of time-independence in previous research.

In this paper, we study the proactive scheduling problem on a generalized uncertainty model, which covers both the traditional stochastic RCPSP and the *time-dependent workability uncertainty*. Our approach adopts the minimization of expected makespan as the optimization objective, and produces Partial-order Schedules (POS) (Policella, Smith, Cesta, & Oddi, 2004) as proactive solutions. Compared to start-time schedules, POS is more flexible in handling unforeseen events, since activity start times are determined online based on actual durations (Fu, Lau, Varakantham, & Xiao, 2012). However, it is not trivial to find the optimal solution for our problem, not only due to the well-known intractability of RCPSP (Blazewicz, Lenstra, & Kan, 1983). Handling uncertainties is challenging, since even evaluating a solution is intractable. Hagstrom (1988) showed that for stochastic RCPSP without resource constraints, complexity of computing expected makespan is #P-complete.

We mitigate the complexity resulted from the stochasticity based on Sample Average Approximation (SAA) (Kleywegt, Shapiro, & Homem-de Mello, 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, then optimally solves the resulting SAA problem. As will be shown later, the SAA problem is NP-hard due to the combinatorial structure of RCPSP. Therefore, we design two branch-and-bound algorithms to solve the SAA problem optimally by exploiting interesting problem structures. These two algorithms differ in the way of how the POS is constructed. More specifically, the first algorithm constructs a POS by linking the activities one by one using feasible resource flows. The second algorithm finds a POS by iteratively resolving possible resource conflicts represented as Minimal Critical Sets (MCS) (Laborie, 2005) using additional precedence constraints. To show the effectiveness of our approach, we evaluate the two algorithms on multiple uncertainty models from existing works and real-world data, and compare the solution quality with the best approaches available.

This paper is an extension of our previous work (Song, Kang, Zhang, & Xi, 2017b), where the flow-based branch-and-bound algorithm is proposed for solving the proactive scheduling problems only with the workability uncertainty. Considerable improvements have been made in this version, including:

- We generalize the activity duration model presented by Song et al. (2017b) to incorporate the traditional stochastic RCPSP model, such that our approach can be applied to solve uncertainty models containing the time-independent components.
- We design another branch-and-bound algorithm, i.e. the MCS-based algorithm, which finds a POS from a different angle than the flow-based algorithm. In addition, we adopt the constraint propagation procedure presented by Song, Kang, Zhang, and Xi (2018) to speed up the MCS-based searching process. As will be shown in the experimental results, the MCS-based algorithm with constraint propagation performs significantly better in terms of computational efficiency than the flow-based algorithm.
- We conduct additional empirical analysis to examine the performance of our approach, including experiments on pure stochastic RCPSP models and mixture models with both time-independent and time-dependent components.

The rest of this paper is organized as follows. Section 2 gives a review of existing works that are related to our problem. In Section 3, we provide the basic notations and important concepts that will be used in this paper. In Section 4, we formulate our proactive scheduling problem and show how to approximate it using SAA. In Section 5, we study some interesting properties of the SAA problem, which will be used in designing our algorithms. Section 6 is devoted to the detailed description of the two branch-and-bound algorithms. Finally, we present the empirical results on benchmark problem instances and different distributions in Section 7, and conclude the paper in Section 8.

2. Related Work

Considerable amount of works have been done for planning and scheduling under uncertainty. Several surveys (Bruni et al., 2015; Bidot et al., 2009; Herroelen & 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 the type of solutions generated, following the taxonomy established by Bidot et al. (2009).

The first category of approaches adopts start-time schedules as solutions, where the start time of each activity is fixed. 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 & Demeulemeester, 2016; Varakantham, Fu, & Lau, 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. However, 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 schedule, a flexible solution does not directly determines the start time of each activity. Instead, the complete schedule is determined along with execution, based on the flexible solution and the actual execution situations. A typical example of flexible solution is dynamically controllable Simple Temporal Network with Uncertainty (STNU) (Cui, Yu, Fang, Haslum, & Williams, 2015; Morris, Muscettola, & Vidal, 2001; Morris & Muscettola, 2005). However, STNU-based approaches are not directly applicable to resource-constrained settings, since they often focus on temporal reasoning only. Redundancy-based techniques for machine breakdowns (Davenport, Gefflot, & Beck, 2001; Lambrechts, Demeulemeester, & Herroelen, 2011) are another type of approaches that generate flexible solutions by protecting activities using extra temporal slacks. More specifically, the durations of some activities which could be affected by the machine breakdowns will be elongated in a proactive way, such that the impact of machine breakdowns can be absorbed. 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, for example normal (Davenport et al., 2001) and exponential distributions (Lambrechts et al., 2011). In contrast, our approach is built on principled approximation scheme SAA, and does not require the stochastic knowledge to follow certain distributions thanks to the sampling procedure.

For RCPSP, perhaps the most commonly used type of flexible solution is Partial-order Schedule (POS). Essentially, a POS adds a set of additional precedence constraints between activities to resolve any possible resource conflict during execution. Policella et al. (2004) proposed two approaches to directly generate POS from a deterministic RCPSP instance, including (a) Envelop Based Algorithm (EBA) and (b) Earliest Start Time Algorithm (ESTA). EBA generates a POS by reasoning with the resource contention represented as the resource envelope (Muscettola, 2002), while ESTA first constructs a start-time schedule and then transforms it to a POS using a chaining procedure. More details about these two approaches will be provided in Section 7.1. 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, as will be shown in our experiments, 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 & Wilson, 2007; Fu et al., 2012; Fu, Lau, & Varakantham, 2015; Fu, Varakantham, & Lau, 2016), for optimizing the risk-aware 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 by Bidot et al. (2009) is conditional schedule, referring to solutions with a set of mutually exclusive decision branches. Decisions of which branches to go will be made at certain time points during execution, contingently based on the actual execution. A typical example of such kind of proactive model is Markov Decision Process based techniques. For the traditional stochastic RCPSP which aims at minimizing the expected makespan on time-independent duration uncertainties, a number of approaches have been proposed to optimize the so-called elementary policies, which are essentially conditional schedules (Igelmund & Radermacher, 1983; Möhring, 2000; Stork, 2001; Ballestín, 2007; Ashtiani, Leus, & Aryanezhad, 2011; Creemers, 2015). The current best approach for stochastic RCPSP is the dynamic programming procedure proposed by Creemers (2015). Our proactive approaches differ from this work in three 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. But the resulting precedence network of an elementary policy is not necessarily a POS, since the resource conflicts can only be resolved for the actual scenario, instead of for all the possible scenarios. Hence, POS can be viewed as a special case of elementary policy, therefore 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 conditional schedules are very high (Bidot et al., 2009). Secondly, the proposed dynamic programming procedure can only terminate when the optimal policy is found, while our algorithms can be terminated anytime with high-quality feasible solutions. Finally, this approach can only solve stochastic RCPSP with time-independent uncertainties, while our approach is applicable to the time-dependent workability uncertainty.

3. Preliminaries

In this section, we introduce the basic notations used in this paper, and some important concepts that are closely related to our approaches.

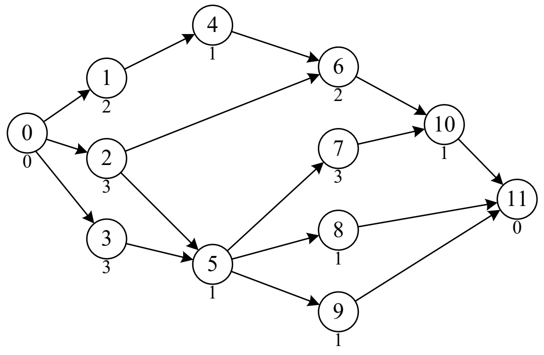


Figure 1: An example of the AON network

3.1 Deterministic RCPSP

An instance of deterministic RCPSP involves a set of activities $A = \{a_1, ..., a_N\}$ that need 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. Each activity $a_i \in A$ has a fixed duration of $d_i^s \in \mathbb{N}$ time slots, and requires $b_{ik} \in \mathbb{N}$ units of resource r_k. We follow the common assumption of non-preemptive activities, i.e. an activity cannot be interrupted once started. Usually two dummy activities a_0 and a_{N+1} with zero durations and no resource requirement 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. Then, the temporal constraints within a project can be represented as an Activity-On-Node (AON) network, which is a directed acyclic graph (DAG) $G_p = (A_p, E_p)$. The AON network of a sample project containing 10 activities is shown in Figure 1, where a circle represent an activity, and the number below it is the requirement for a single resource with limited capacity of 4 units. In addition, the arrows between circles represent the precedence relations. Throughout this paper, we 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.

A solution to a deterministic RCPSP 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. For an activity a_i in the deterministic RCPSP, once s_i is determined, its completion time $c_i = s_i + d_i^s$ is also determined. A feasible schedule must satisfy all the resource and precedence constraints. A feasible schedule S^* is optimal if it minimizes the makespan $MS(S) = \max_i\{c_i\} = c_{N+1}$.

3.2 Partial-order Schedule and AON-flow Network

Under uncertain activity durations, a feasible schedule S generated by solving a deterministic instance could become infeasible during execution, since the activity completion times are also uncertain. Whenever this happens, the schedule must be repaired (i.e. restore the

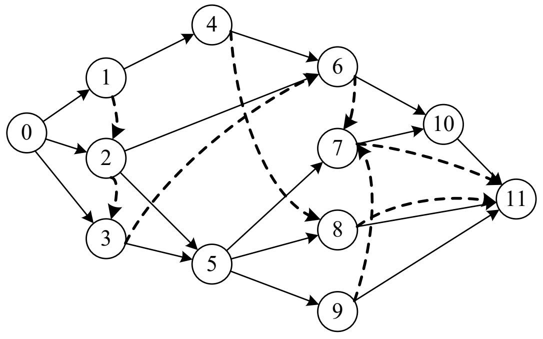


Figure 2: An example of POS

feasibility) so that the project execution can be resumed. The repair procedures are often expensive, since both temporal and resource constraints need to be taken into account.

Different from the start-time schedules, Partial-order Schedule (POS) (Policella et al., 2004) is a type of flexible solution for proactive scheduling problems, where the start time of each activity is determined during execution time instead of before execution. A POS is a graph $G_R = (A_p, E_p \cup E_R)$ 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 in Figure 1 is shown in Figure 2, where the dotted arrows represent the additional precedence constraints in E_R. POS provides a very efficient way for dealing with uncertain durations, by avoiding the requirement of reasoning for complex resource constraints. Specifically, the start time of each activity is determined solely based on the precedence constraints and realized 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:

$$s_i = \max\{c_j = s_j + d_j | a_j \in Pre(a_i, G_R)\}, \tag{1}$$

where d_j is the realized duration of a_j. In other words, a_i is started after the completion of all its immediate predecessors. Along with execution, the start times of all $a_i \in A_p$ can be determined, hence a feasible schedule is obtained. We refer to the set of all POS as $\boldsymbol{G}_R$.

Another concept that will be used here is the AON-flow Network (Artigues, Michelon, & Reusser, 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 & 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 to be

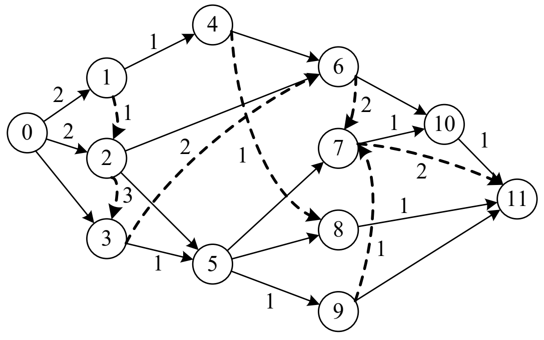


Figure 3: An example of AON-flow Network

$\emptyset$, which means some edges in the original AON network E_p may also carry resource flows. Figure 3 shows a sample AON-flow Network for the project in Figure 1, where the arrows associated with values represent the resource carrying edges in E_F, and the values indicate the transfered resource units.

Artigues et al. (2003) mentioned that, 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 6.1.

4. Problem Formulation

In this section, we first model the uncertainty in Section 4.1, and then formulate the stochastic optimization model of our proactive problem and its SAA approximation in Section 4.2.

4.1 The Model of Uncertainty

We first briefly describe the workability uncertainty model introduced by Song et al. (2017b). Without loss of generality, we first classify all activities in A into Z types, such that activities

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

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 (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 with 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. Such uncertainty model may exist in many applications. Essentially, it describes the random workability (when $\boldsymbol{X}_{zt}$ is binary) or working amount (when $\boldsymbol{X}_{zt}$ is non-binary) of an activity on each time slot. One type of applications is the motivating example we described in the introduction, where the activity may not be executed on certain time slots if the weather condition is not satisfied. Another type of related scenarios is in the transportation or vehicle routing applications (Cao, Guo, Zhang, Oliehoek, & Fastenrath, 2017; Cao, Guo, Zhang, Niyato, & Fastenrath, 2016), where the distance that can be traveled or goods that can be delivered by a vehicle on each specific time period is random. It should be noted that though the random matrix $\boldsymbol{X}$ is defined on all t, in many applications the model does not need to cover the whole horizon due to the periodicity of uncertainty patterns (e.g. daily, weekly, yearly, etc.).

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. As Song et al. (2017b), 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:

$$\begin{aligned} F_{\boldsymbol{D}_i}\left(d_i \mid s_i = t\right) &= P\left(\boldsymbol{D}_i \le d_i \mid s_i = t\right) \\ &= P\left(\sum_{\tau=t}^{t+d_i-1} \boldsymbol{X}_{z_i\tau} \ge d_i^s \middle| s_i = t\right), \end{aligned} \tag{2}$$

where d_i is a possible 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 duration model in Equation (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 its start time s_i. In this case, 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. For example, suppose the execution of an activity is affected by both the weather conditions and unforeseen resource breakdowns, which can be modeled as a time-independent random elongation to the deterministic activity duration (Fu et al., 2015; Lambrechts et al., 2011). Note that the two uncertainty sources affect the activity duration in different ways: the random resource breakdown determines how many workable

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.

time slots are needed (i.e. the baseline duration), while the weather condition determines whether a time slot can be counted as workable. 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 formulated as:

$$\begin{aligned} F_{\boldsymbol{D}_i}\left(d_i \mid s_i = t, \boldsymbol{Y}_i = y_i\right) &= P\left(\boldsymbol{D}_i \le d_i \mid s_i = t, \boldsymbol{Y}_i = y_i\right) \\ &= P\left(\sum_{\tau=t}^{t+d_i-1} \boldsymbol{X}_{z_i\tau} \ge y_i \middle| s_i = t, \boldsymbol{Y}_i = y_i\right), \end{aligned} \tag{3}$$

where y_i is a possible realization of $\boldsymbol{Y}_i$. We denote the uncertainty model studied in this paper 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 activity a_i, Equation (3) is equivalent to Equation (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 1. *When $P(\boldsymbol{X}_{zt} = 1) = 1$ holds for all z and t, the random duration $\boldsymbol{D}_i$ specified by Equation (3) is time-independent, and shares the same distribution with $\boldsymbol{Y}_i$.*

The correctness of Observation 1 can be verified as follows:

$$\begin{aligned} F_{\boldsymbol{D}_i}\left(d_i \mid s_i = t, \boldsymbol{Y}_i = y_i\right) &= P\left(\boldsymbol{D}_i \le d_i \mid s_i = t, \boldsymbol{Y}_i = y_i\right) \\ &= P\left(\sum_{\tau=t}^{t+d_i-1} \boldsymbol{X}_{z_i\tau} \ge y_i \middle| s_i = t, \boldsymbol{Y}_i = y_i\right) \\ &= P\left(d_i \ge y_i | s_i = t, \boldsymbol{Y}_i = y_i\right) = P(\boldsymbol{Y}_i \le d_i). \end{aligned} \tag{4}$$

Therefore, the uncertainty model $\boldsymbol{U} =< \boldsymbol{X}, \boldsymbol{Y} >$ and the CDF in Equation (3) generalize both the traditional time-independent uncertainty and the time-dependent workability uncertainty at the same time. Note that model $\boldsymbol{U}$ is a special case of the more general time-dependent uncertainty model. It is an important future direction for us to explore how to efficiently handle the general time-dependent uncertain durations.

4.2 The Proactive Problem and Sample Average Approximation

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

$$G_R^* = \underset{G_R \in \boldsymbol{G}_R}{\operatorname{argmin}} \left\{ g(G_R) = \mathbb{E}[MS(G_R, \boldsymbol{U})] \right\}, \tag{5}$$

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 in the above formulation and the algorithms that will be presented in the following sections, we make the assumption that the online decision making is purely based on Equation (1). If more complex online procedures are allowed, then it must be explicitly incorporated in Equation (6) such that the proactive scheduling problem is meaningful. Though this could lead to better expected makespan, the proactive algorithm must be designed based on detailed analysis on the specific online procedure.

Equation (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, which is shown to be #P-complete (Hagstrom, 1988). 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, here we use Sample Average Approximation (SAA) to approximate the problem in Equation (5). 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 (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 (5) can be formulated as:

$$\hat{G}_R^* = \underset{G_R \in \boldsymbol{G}_R}{\operatorname{argmin}} \left\{ \hat{g}(G_R) = \frac{1}{Q} \sum_{q=1}^{Q} MS(G_R, u^q) \right\}, \tag{6}$$

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 (5), and $MS(G_R, u^q)$ is the makespan of a solution G_R on sample u^q. As proved by Kleywegt et al. (2002), $\hat{G}_R^*$ will converge to G_R^* at an exponential rate with the increase of Q. Meanwhile, Equation (6) is a deterministic problem instead of a stochastic one, which can help to avoid reasoning on the complex uncertainty models and facilitate the design of algorithms. More importantly, in the next section, we will show that the complexity of computing $\hat{g}(G_R)$ for a given G_R is polynomial-time, which significantly simplifies the hardness of solution evaluation.

5. Properties of the SAA Problem

In this section, we study some interesting properties of the SAA problem in Equation (6). First, 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 (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. However, the duration of each activity is not directly specified in u. Below we analyze how to compute such duration.

Intuitively, the duration of an activity on a sample should be 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:

$$d_i(s_i, u) = \min \left\{ d > 0 \middle| \sum_{\tau=s_i}^{s_i+d-1} x(u, z_i, \tau) \geq y(u, i) \right\}. \tag{7}$$

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 show that an activity can never finish earlier by starting later, by proving the following lemma:

Lemma 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 (7),

$$\sum_{t=s_i^1}^{c_i(1)-1} x(u, z_i, t) = \sum_{t=s_i^2}^{c_i(2)-1} x(u, z_i, t) = y(u, i). \tag{8}$$

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

$$\sum_{t=s_i^1}^{s_i^2-1} x(u, z_i, t) + \sum_{t=s_i^2}^{c_i(2)-1} x(u, z_i, t) + \sum_{t=c_i^2}^{c_i(1)-1} x(u, z_i, t) = y(u, i). \tag{9}$$

Since the second term in the left hand side of Equation (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 (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 (7) which states that $d_i(s_i^1, u)$ is the minimum feasible duration. □

According to Equations (1) and (7), we can now obtain a start time schedule of a POS G_R on a given sample u. We denote this schedule as $S(G_R, u)$. Then, based on Lemma 1, we can show that the schedule obtained in such way is sufficient for our purpose of minimizing makespan by proving the following proposition:

Proposition 1. *Given a POS G_R and a sample u, the schedule $S(G_R, u)$ produces the lowest makespan.*

Algorithm 1: ComputeMakespan(G_R, u)

```
   Input: $G_R$: a solution; $u$: a sample
   Output: $MS(G_R, u)$: the makespan of $G_R$ on $u$
1  $CS \leftarrow \{a_0\}$, $ES \leftarrow \{a_i \notin CS | Pre(a_i) \subseteq CS\}$ ;
2  while $|CS| < |A_p|$ do
3      foreach $a_i \in ES$ do
4          $s_i \leftarrow \max\{c_j | a_j \in Pre(a_i)\}$ ;
5          $c_i \leftarrow s_i + d_i(s_i, u)$ ;
6          $CS \leftarrow CS \cup \{a_i\}$ ;
7      $ES \leftarrow \{a_i \notin CS | Pre(a_i) \subseteq CS\}$;
8  return $MS(G_R, u)$ ;
```

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 (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 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 (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 1 and Proposition 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. Meanwhile, we can have the following observation which will be used in designing our algorithms:

Observation 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 (1). This indicates that a_i cannot complete earlier by using G_R^2 than using G_R^1, according to Lemma 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 1, we give a simple algorithm with a complexity of $\mathcal{O}(N^2T)$. Therefore, it is tractable to evaluate the objective $\hat{g}(G_R)$, with a complexity of $\mathcal{O}(MN^2T)$. However, the SAA problem is intractable, as stated below:

Proposition 2. *The SAA problem in Equation (6) is NP-hard.*

Proof. The NP-hardness can be shown through reduction from the decision version of deterministic RCPSP, which is well-known to be NP-complete. Specifically, for a given deterministic RCPSP instance, we can construct an instance of the SAA problem, by adding one sample u where $x_{zt} = 1$ for all z and t, and $y_i = d_i^s$ for each activity a_i with d_i^s being the deterministic duration of a_i.

We claim that the SAA problem has a solution G_R with $\hat{g}(G_R) \leq t$ if and only if there is a schedule S for the deterministic RCPSP instance with $MS(S) \leq t$. If such a schedule S exists, then a POS G_R can be immediately obtained by the chaining procedure (Policella, Cesta, Oddi, & Smith, 2009) in polynomial time. 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 $MS(G_R, u) \leq t$. $\square$

6. Branch-and-Bound Algorithms

Since the SAA problem is intractable, in this section, we design two branch-and-bound algorithms to solve it efficiently. We begin with analyzing the relations between POS and AON-flow Network. Then, we propose two branch-and-bound algorithms, namely the flow-based algorithm and the MCS-based algorithm.

6.1 Relations between POS and AON-flow Network

As we have mentioned in Section 3.2, 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 show that given a POS, such an AON-flow Network indeed exist, and can be found in polynomial time using maximum flow algorithms.

First, for a given DAG $G = (A_p, E)$, we introduce a method for determining whether an AON-flow Network G_F with $E(G_F) \subseteq E(G)$ exist. When there is only one resource r with capacity C, it has been shown by Leus and Herroelen (2004) that the existence of such an AON-flow Network 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. 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).

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 $r_k \in R$. Based on this

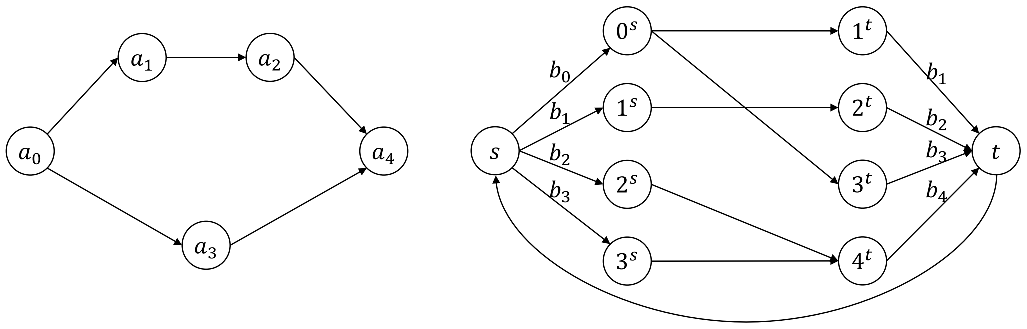


Figure 4: An Example of Network Transformation (left: original DAG G with $N = 3$; right: transformed network G^T, where b_i beside edges represent capacities)

procedure, we can show that given a POS, we can find an AON-flow Network that uses the same set or a subset of the precedence constraints in polynomial time:

Proposition 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 additional precedence constraints are needed to resolve resource conflicts. □

Proposition 3 indicates that a POS G_R can "accommodate" at least one AON-flow Network. In addition, it 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]:

$$\hat{G}_F^* = \underset{G_F \in \boldsymbol{G}_F}{\operatorname{argmin}} \left\{ \hat{g}(G_F) = \frac{1}{Q} \sum_{q=1}^{Q} MS(G_F, u^q) \right\}. \tag{10}$$

This can be justified by the following conclusion:

Proposition 4. *For any optimal solution $\hat{G}_F^*$ of the problem in Equation (10), the corresponding POS $\hat{G}_R^* = (A_p, E(\hat{G}_F^*))$ solves the problem in Equation (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 3, there must exist an AON-flow Network G'_F with $E(G'_F) \subseteq E(G'_R)$. Based on Observation 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. □

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

Next, we introduce our two branch-and-bound algorithms.

6.2 The Flow-based Algorithm

Our first algorithm, named `BnB-Flow`, searches for the optimal AON-flow Network to obtain the optimal POS. This is done by linking each activity to a partial solution using feasible flows, which will be detailed as follows.

6.2.1 Branching Scheme

Essentially, `BnB-Flow` is a depth-first tree search process that 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 added to the partial solution, along with the corresponding edge set that we call it a *link*. 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 (a set of edges) will be selected for branching.

The branch-and-bound process can be described by the pseudo code in Algorithm 2, 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 6.2.4. Once a_l is chosen, the algorithm computes the lower bound of this branching choice (adding 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 6.2.2. 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_Flow 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

Algorithm 2: BnB_Flow($G'_F, \hat{G}^*_F, \hat{g}^*, OC, \boldsymbol{u}$)

```
Input: $G'_F$: current partial solution; $\hat{G}^*_F$: current best solution; $\hat{g}^*$: current best
       objective value; $OC$: outgoing capacity matrix; $\boldsymbol{u}$: the sample set
1  $ES \leftarrow$ FindEligibleActivities($V(G'_F)$) ;
2  if $ES = \emptyset$ then
3      $\hat{g}' \leftarrow$ComputeObj($G'_F, \boldsymbol{u}$) ;
4      if $\hat{g}' < \hat{g}^*$ then
5          $\hat{g}^* \leftarrow \hat{g}'$;
6          $\hat{G}^*_F \leftarrow G'_F$;
7      return;
8  while $ES \neq \emptyset$ do
9      $a_l \leftarrow$ ChooseActivity($ES$);
10     $ES \leftarrow ES \setminus \{a_l\}$;
11     $LB(G'_F, a_l) \leftarrow$ ComputeLB_A($G'_F, a_l, \boldsymbol{u}$);
12     if $LB(G'_F, a_l) < \hat{g}^*$ then
13         $LK \leftarrow$ FindFeasibleLinks($G'_F, a_l, OC$);
14         $lk \leftarrow$ChooseLink($G'_F, LK, \hat{g}^*, \boldsymbol{u}$);
15         while $lk \neq null$ do
16             $LK \leftarrow LK \setminus \{lk\}$;
17             $\bar{G}'_F \leftarrow$LinkActivity($a_l, G'_F, lk, OC$);
18             BnB_Flow($\bar{G}'_F, \hat{G}^*_F, \hat{g}^*, OC, \boldsymbol{u}$);
19             $G'_F \leftarrow$RemoveActivity($a_l, \bar{G}'_F, lk, OC$);
20             $lk \leftarrow$ChooseLink($G'_F, LK, \hat{g}^*, \boldsymbol{u}$);
21 return;
```

bounds for the link level are embedded in the ChooseLink function shown in Algorithm 3, and will be discussed in Section 6.2.2.

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 2, if a_l is linked to G'_F using a link lk, then for each $r_k \in R$ and $a_l \in V(lk)$, 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($G'^0_F, null, L, OC^0, \boldsymbol{u}$), where $G'^0_F = (\{0\}, \emptyset)$ is the initial 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.

6.2.2 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 $\bar{G}'_F = (V(G'_F) \cup \{a_l\}, E(G'_F) \cup lk)$:

$$(a_i, a_l) \in Tr(\bar{G}'_F), \forall a_i \in Pre(a_l) \tag{11}$$

$$\sum_{(a_i,a_l)\in lk} f_{ilk} = b_l^k, \forall r_k \in R. \tag{12}$$

The first condition in Equation (11) guarantees the original precedence constraints in G_p is respected by $\bar{G}'_F$. The second condition in Equation (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 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 3, there must be an AON-flow Network G'_F with $E(G'_F) \subseteq E(G_R)$. As we mentioned in Section 6.1, the flow values in G'_F should be integers. Based on the same procedure in the proof of Proposition 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, for each resource r_k that a_l has a positive requirement (i.e. $b_{ik} > 0$), all activities $a_i \in G'_F$ with positive oc_{ik} values are identified as the candidates for linking a_l. Then, we find all the feasible resource transfer combinations (i.e. a vector of resource flows $f_{ilk} \in [0, oc_{ik}]$) that satisfy the condition in Equation (12) and contain only positive resource flows (i.e. $\sum_{r_k \in R} f_{ilk} > 0$), which form a set LK. Finally, each element $lk \in LK$ is checked against the condition in Equation (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.

The branching and pruning process in the link level is shown in Algorithm 3. When LK is not empty, an element is selected based on certain criterion (will be further discussed in Section 6.2.4) 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 2 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 3 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:

```
Algorithm 3: ChooseLink($G'_F, LK, \hat{g}^*, \boldsymbol{u}$)
  Input: $G'_F$: current partial solution; $LK$: current set of feasible links; $\hat{g}^*$: current
         best objective value; $\boldsymbol{u}$: the sample set
  Output: $lk$: the chosen link
1 while $LK \neq \emptyset$ do
2     $lk \leftarrow$GetLink($LK$);
3     $LB(G'_F, lk) \leftarrow$ComputeLB_L($G'_F, lk, \boldsymbol{u}$);
4     if $LB(G'_F, lk) < \hat{g}^*$ then
5         return $lk$;
6     else
7         $LK \leftarrow$RemoveLinks($LK, lk$);
8 return null;
```

Observation 4. *In Algorithm 3 where ComputeLB_L is used for lower bound computation, 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 will be justified in Section 6.2.3 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 6.2.3 does.

6.2.3 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 (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 below equation is an admissible lower bound of $\hat{g}$:

$$LB(G'_F, \Delta) = \frac{1}{Q}\sum_{q=1}^{Q} MS_{LB}(G'_F, \Delta, u^q), \tag{13}$$

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 (13) and Observation 2, we construct the two lower bounds based on the critical path lower bound for solving deterministic RCPSP (Demeulemeester & 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:

$$LB(G'_F, lk) = \frac{1}{Q}\sum_{q=1}^{Q} MS(\bar{G}''_F, u^q). \tag{14}$$

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 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 (13).

Now we show the the correctness of Observation 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 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. Basically, all activities are split into three parts, namely those in G'_F, a_l it self, and those not in G'_F except a_l, denoted as $A^r_p = A_p \setminus (V(G'_F) \cup \{a_l\})$. For each sample u, we aim at obtaining a precedence feasible (but may not be resource feasible) schedule $S(G'_F, a_l, u)$ by determining the start times of the activities in the above three parts separately, such that $MS(G'_F, a_l, u) = MS(S(G'_F, a_l, u))$ is a lower bound for the branching decision (linking a_l to G'_F) on u. To achieve this, we first propagate G'_F on u to obtain a schedule $S(G'_F, u)$ that contains only the current activities in G'_F. Then, based on $S(G'_F, u)$, we compute the earliest precedence and resource feasible start time s_l of the chosen activity a_l, along with the duration $d_l(s_l, u)$ computed using Equation (7). Finally, based on $S(G'_F, u)$ and s_l, we obtain the complete schedule $S(G'_F, a_l, u)$ by determining the start times for all the activities in A^r_p, i.e. those not in G'_F except a_l, according to only the original precedence constraints.

Below we show that $MS(G'_F, a_l, u)$ is indeed a lower bound for the makespan of the current branching decision on u. Let $\bar{G}'_F$ be a partial solution obtained by linking a_l to G'_F using a feasible link. Then clearly for any $G_F \in \boldsymbol{G}_F$ obtained by extending any $\bar{G}'_F$, s_l is the earliest possible start time for a_l on u. According to Lemma 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:

$$LB(G'_F, a_l) = \frac{1}{Q}\sum_{q=1}^{Q} MS(G'_F, a_l, u^q). \tag{15}$$

6.2.4 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 2, 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 3, 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 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, which prefers a link with a smaller number of vertices, i.e. $|V(lk)|$.

Discussion. Our algorithm BnB-Flow is conceptually similar to the precedence tree based algorithms developed for deterministic RCPSP (Patterson, Talbot, Slowinski, & Węgłarz, 1990; Sprecher & Drexl, 1998). However, we need to emphasize that our problem in Equation (10) differs from deterministic RCPSP in that we need to construct a *common* AON-flow Network that produces the lowest average makespan on *multiple samples*. This leads to two differences in algorithm design. Firstly, to find an AON-flow Network, we need the second level in the branching scheme to decide how an eligible activity is connected to a partial network. In contrast, for deterministic RCPSP, one level branching is sufficient since there is no need to consider any additional precedence and resource transfer relations. Secondly, due to the existence of multiple samples, the dominance rules (e.g. swapping rule, local/global left shift rule) designed for deterministic RCPSP cannot be directly applied. Nevertheless, we plan to study how to introduce effective dominance rules to BnB-Flow, since they could help in significantly improving the computational efficiency.

6.3 The MCS-based Algorithm

The Flow-based algorithm searches for the optimal POS by searching for the optimal AON-flow Network. However, the space of AON-flow Networks may be much larger than that of POS, due to the one-to-many relation between POS and AON-flow Networks. Intuitively, for an AON-flow Network G_F, there is a POS $G_R = (V(G_F), E(G_F))$ with the same SAA objective value, i.e. $\hat{g}(G_F) = \hat{g}(G_R)$. However, there could be a number of $G'_F \in \boldsymbol{G}_F$ with $E(G'_F) = E(G_F)$. Though the flow values are different, these AON-flow Networks have the same SAA objective value as G_F and G_R, since the flow values are not used in computing

$\hat{g}$. Therefore, search in the space of AON-flow Networks could be less efficient. Different from `BnB-Flow`, our second algorithm, named `BnB-MCS`, directly searches for the optimal solution in the space of POS. More specifically, starting from the original AON network, this algorithm employs an iterative process of detecting possible resource conflicts represented as *Minimal Critical Set* (MCS), and resolving them by adding precedence constraints. When no MCS can be detected, a POS is found. Note that for a detected MCS, there could be multiple ways to resolve it, i.e. adding a precedence constraint between different activity pairs. Based on this fact, `BnB-MCS` systematically searches for the best way of constructing POS by trying different alternatives of MCS resolution as branching candidates, which will lead to the optimal solution of Equation (6).

6.3.1 Detecting and Resolving Minimal Critical Sets

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

Definition 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 2. *If a critical set A_{mc} satisfies $\forall a_i \in A_{mc}$, $\sum_{a_j \in A_{mc} \setminus \{a_i\}} b_j^k \leq C_k$, then it is a Minimal Critical Set (MCS).*

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, a 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 by 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[4] $G^M(k)$, which can be solved in polynomial time using the inverse Ford-Fulkerson's algorithm (Lombardi & 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

4. Note that this network transformation is different from the one we described in Section 6.1. Here we ignore the details of this transformation procedure for brevity.

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 & Milano, 2012; Lombardi, Milano, & Benini, 2013)[5], where temporal reasoning can be applied for branching and constraint propagation. In contrast, our problem in Equation (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.

6.3.2 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 adding precedence constraints to G_p to resolve MCS, until a conflict-free augmented DAG is obtained. When a MCS is detected, `BnB-MCS` will try all the possible ways (i.e. adding precedence constraint) to resolve it, which are considered as branching alternatives. For MCS detection, we adopt the method proposed by 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 6.3.4. In addition, we extend the constraint propagation procedure proposed by 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 $0 \le f^L_{ijk} \le f^U_{ijk}$. 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 6.3.3.

Detail of this branching process is shown in the BnB_MCS function in Algorithm 4. Inputs of the algorithm include 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 the sample set $\boldsymbol{u}$. The first operation in Algorithm 4 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 $E(G_R) \subseteq E(G'_R)$, $MS(G_R, u) \le MS(G'_R, u)$ holds for any sample u according to Observation 2, therefore $\hat{g}(G_R) \le \hat{g}(G'_R)$ 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).

5. 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.

Algorithm 4: BnB_MCS($G'_R, \hat{G}^*_R, \hat{g}^*, \boldsymbol{u}$)

```
Input: $G'_R$: current partial solution; $\hat{G}^*_R$: current best solution; $\hat{g}^*$: current best
       objective value; $\boldsymbol{u}$: the sample set
1  $A_{mc} \leftarrow$ DetectMCS($G'_R$) ;
2  if $A_{mc} = \emptyset$ then
3      $\hat{g}' \leftarrow$ComputeObj($G'_R, \boldsymbol{u}$) ;
4      if $\hat{g}' < \hat{g}^*$ then
5          $\hat{g}^* \leftarrow \hat{g}'$;
6          $\hat{G}^*_R \leftarrow G'_R$;
7      return;
8  else
9      $Res^L(A_{mc}) \leftarrow$GetRankedResolvers($A_{mc}$) ;
10     foreach $(a_i, a_j) \in Res^L(A_{mc})$ do
11         if $sum^U_{ij} = 0$ then
12             continue;
13         if ComputeLB_MCS($G'_R, a_i, a_j, \boldsymbol{u}$)$< \hat{g}^*$ then
14             $G'_R \leftarrow (A_p, E(G'_R) \cup \{a_i, a_j\})$;
15             $R^L \leftarrow$GetRankedResources($a_i, a_j$);
16             for $r_k \in R^L$ do
17                 if propagateLB($a_i, a_j, k$)=true then
18                     BnB_MCS($G'_R, \hat{G}^*_R, \hat{g}^*, \boldsymbol{u}$);
19                     Restore();
20             $G'_R \leftarrow (A_p, E(G'_R) \setminus \{(a_i, a_j)\})$;
21         $f^U_{ijk} \leftarrow 0, f^L_{ijk} \leftarrow 0$ for all $r_k \in R$;
22         if propagateUB($a_i, a_j$)=true then
23             BnB_MCS($G'_R, \hat{G}^*_R, \hat{g}^*, \boldsymbol{u}$);
24             Restore();
25 return;
```

In Lines 10-24, the ranked resolvers are selected for branching one by one. For a selected resolver, the algorithm first checks if it is *applicable*, i.e. not being banned by the current branching decisions (Lines 11-12). 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 compares 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 be detailed in Section 6.3.4. 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 tested, 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. imposing $f^U_{ijk} = 0$ (which automatically imposes $f^L_{ijk} = 0$) for all $r_k \in R$ (Line 21), and propagate it to maintain bound consistency (Line 22). If the propagation is successful, the algorithm continues by calling BnB_MCS. The algorithm is invoked by calling BnB_MCS$(G_p, null, L, \boldsymbol{u})$. Upon termination, the optimal POS can be found, if the lower bound is admissible.

Discussion. In a previous work (Song et al., 2018), we design a branch-and-bound algorithm to search for the optimal POS of the risk-aware proactive scheduling problem. Here we adopt the same constraint propagation procedure, but use a different branching scheme. In the algorithm proposed by Song et al. (2018), given a partial solution G'_R, the next edge for branching is directly selected from the set $FS(G'_R)$ containing all the edges that can be added to G'_R, based on the ranking heuristic. Though this procedure does not affect the correctness of the algorithm, the chosen edge may not resolve any resource conflict. We call this algorithm `BnB-Edge`. In `BnB-MCS`, the next branching candidate is selected from a subset of $FS(G'_R)$, i.e. the applicable resolvers of a detected MCS, therefore adding this candidate can indeed help in reducing resource conflicts. Furthermore, these resolvers will be at the same branching level, i.e. the algorithm will try each of them to resolve the same MCS. Though this would lead to a search tree with larger width, the depth may be smaller since the edges are organized in a meaningful way by the detected MCS. In our experiments, we will show that `BnB-MCS` exhibits better performance than `BnB-Edge`.

6.3.3 Constraint Propagation

In this section, we present our constraint propagation method in detail. For single resource problems, Leus and Herroelen (2004) proposed to maintain the flow bound consistency by conducting constraint propagation on the *remainder network* $G_{RD} = (A_p, E_p \cup E_{RD})$, where $E_{RD} = \{(a_i, a_j) \in FS | f^U_{ij} > 0\}$ 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:

$$f^L_{ij} = \max\left\{ f^L_{ij}, b_i - \sum_{(a_i,a_l)\in OT_{ij}} f^U_{il}, b_j - \sum_{(a_l,a_j)\in IN_{ij}} f^U_{jl} \right\} \tag{16}$$

$$f^U_{ij} = \min\left\{ f^U_{ij}, b_i - \sum_{(a_i,a_l)\in OT_{ij}} f^L_{il}, b_j - \sum_{(a_l,a_j)\in IN_{ij}} f^L_{jl} \right\} \tag{17}$$

Consistency can be achieved by updating bounds for all edges in $E(G_{RD})$ till no bound changes. The network G^T_{RD} transformed from G_{RD} using the procedure in Section 6.1 is also used for detecting infeasibility (Leus & Herroelen, 2004). If $f(G^T_{RD}) < f_{max}$, then clearly the current branching decisions cannot lead to any AON-flow Network, hence no POS can be generated according to Proposition 3.

For our problem with multiple resources, we maintain the flow bounds independently for each r_k based on Equations (16) and (17). The branching decisions on resources in

Algorithm 4 enable the independent bound updates: when an edge (a_i, a_j) is included, f^L_{ijk} 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^U_{ijk} to 0 (so as f^L_{ijk}) for all r_k and propagating to other bounds. If any bound infeasibility (i.e. $f^U_{ijk} < f^L_{ijk}$) 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 $sum^L_{ij} > 0$) or banned (if $sum^U_{ij} = 0$).

If the flow bounds are updated successfully, propagateLB and propagateUB try to detect flow infeasibility. For each r_k, we maintain the transformed networks $G^T_{RD}(k)$ and $G'^T_R(k)$ for the current remainder network G_{RD} and partial solution G'_R, 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 3, the current branching decisions cannot lead to any POS and a false value is returned to signal backtracking.

6.3.4 Heuristics for CS Reduction and Resolver Selection

The procedure for reduction of CS to MCS proposed in Lombardi and Milano (2012) is based on the so-called preserved space heuristic designed by 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 4) 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:

$$RS(a_i, a_j) = \sum_{r_k \in R} \left\{ RS_k(a_i, a_j) = \frac{f^{RD}_{ijk}}{f^k_{max} - f(G'^T_R(k))} \right\}, \tag{18}$$

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 4). 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^k_c and reduce it to a MCS A^k_{mc} using a greedy procedure iteratively removes an activity from A^k_c that causes the smallest reduction in $RS(A^k_c)$ until a MCS is obtained; 2) return the A^k_{mc} with the maximum $RS(A^k_{mc})$.

Resource score is also used in ranking resolvers and resources for branching. More specifically, function GetRankedResolvers in Line 9 of Algorithm 4 ranks the resolvers in the descending order of their resource score values. Function GetRankedResources in Line 15 of Algorithm 4 ranks all resources also in the descending order, based on the values of $RS_k(a_i, a_j)$ of each resource r_k.

6.3.5 Lower Bound

Here we design an admissible lower bound for `BnB-MCS` following the similar idea in Section 6.2.3, based on Equation (13) and Observation 2. Specifically, in the ComputeLB_MCS function of Algorithm 4 (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 $\hat{g}$.

7. 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 7.1, 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 7.2. Finally, in Section 7.3, we compare our algorithms with several benchmark algorithms on instances with different size and uncertainty models with different configurations.

7.1 Experiment Setting

The RCPSP instances used in our experiments are generated using a widely used benchmark problem generator *RanGen2* (Vanhoucke, Coelho, Debels, Maenhout, & Tavares, 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).

To have a more detailed study of our algorithms, we first generate small sized instances with $N \in \{10, 20, 30\}$. Two sets of instances with $K \in \{1, 2, 3\}$ are generated, with different values of OS, RF and RC. More 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 $d_i^0 \in [1, 10]$. These two instance sets will be used in Section 7.2 and the first

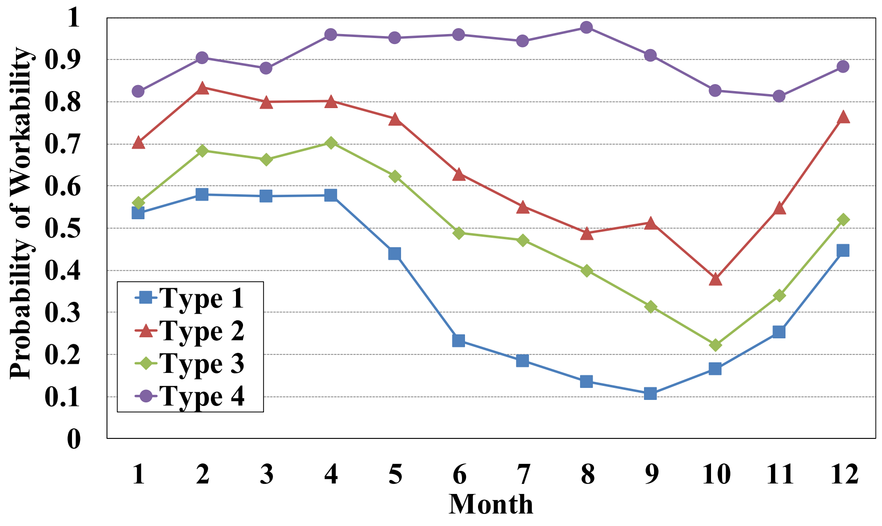


Figure 5: The Trend of Monthly POW

Table 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

three subsections in Section 7.3. To examine the performance on large sized instances, we generate another two sets of instances with $N \in \{60, 120\}$ and the same configurations of K, OS, RF and RC. Results on these instances will be discussed in Section 7.3.4.

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 1 and visualized in Figure 5, 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 7.3.3, we increase the deterministic activity durations d_i^0 of each instance to elongate the critical path 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 by Beck and Wilson (2007); 2) an exponential distribution $\boldsymbol{Y}_i \sim Exp(1/d_i^0)$, which is used by Creemers, Leus and Herroelen (2015, 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 two general-purpose POS generation approaches ESTA-Iter and EBA-Minflow and a randomized local search heuristic RLS. More specifically, general-purpose approaches refer to those that can 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. Details of these benchmark algorithms are listed as follows:

- ESTA-Iter: this algorithm first generates a start-time schedule for the deterministic problem, then transforms it to a POS using a chaining procedure (Policella et al., 2004). We implement the best version of this approach following (Policella et al., 2009), where an iterative chaining procedure (100 iterations) is applied on the schedule generated by the ESTA procedure (Policella, Cesta, Oddi, & Smith, 2007).

- 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 proposed by Lombardi and Milano (2012).

- RLS: we implement a randomized local search algorithm that is similar to the ones used by Fu et al. (2015) and Fu et al. (2012), which can be considered as a heuristic for solving the SAA problem in Equation (6). Essentially, RLS performs randomized search on the neighborhood of an activity list that is compatible with the original precedence constraints in E_p. As shown in Algorithm 5, RLS first generates an initial activity list al randomly (Line 1). Then, up to *maxIter* times of local search iterations are performed (Lines 3-10), where an earliest start schedule ss is generated (Line 4) and then transformed to a POS G'_R using ESTA-Iter (Line 5). The current best solution $\hat{G}^*_R$ will be updated if G'_R has better objective value on the sample set $\boldsymbol{u}$ (Lines 6-9). Finally, a local move will be applied to al, which is to randomly swap two activities in al if the original precedence constraints in E_p is not violated (Line 10). Here we set $maxIter = 1000$, which is also used by Fu et al. (2015) and Fu et al. (2012).

In Section 7.3.3 which presents our results on uncertainty models with only component $\boldsymbol{Y}$, we also compare our algorithms with two benchmark algorithms, including a state-of-the-art solver for time-independent uncertain durations (Creemers, 2015) named as Creemers15, and a simple heuristic named BPS (Best POS in Samples):

- Creemers15: this approach considers the stochastic scheduling procedures as a continuous time Markov Decision Process, and the optimal scheduling policy is found by dynamic programming technique. 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.

- BPS: similar as RLS, BPS is also a heuristic for solving Equation (6). In this algorithm, for each sample $u \in \boldsymbol{u}$, a deterministic RCPSP is constructed by setting the activity

Algorithm 5: Randomized Local Search

```
Input: $G_p$: the original AON network; $\boldsymbol{u}$: the sample set; $maxIter$: the maximum
       number of iterations
Output: $\hat{G}^*_R$: the best solution found
1  $numIter \leftarrow 0, \hat{g}^* \leftarrow \infty$;
2  $al \leftarrow$GenerateActivityList($G_p$);
3  for $numIter \leftarrow 1$ to $maxIter$ do
4      $ss \leftarrow$GenerateSchedule($al$);
5      $G'_R \leftarrow$GetPOS($ss$);
6      $\hat{g}' \leftarrow$ComputeObj($G'_R, \boldsymbol{u}$) ;
7      if $\hat{g}' < \hat{g}^*$ then
8          $\hat{g}^* \leftarrow \hat{g}'$;
9          $\hat{G}^*_F \leftarrow G'_F$;
10     $al \leftarrow$ApplyLocalMove($al$);
11 return $\hat{G}^*_R$;
```

durations to be the sampled values. Then, the RCPSP is solved and the optimal start-time schedule is retrieved and transformed to a POS using ESTA-Iter. Finally, the POS that has the best average makespan on all the samples in $\boldsymbol{u}$ is returned as the solution. We use this benchmark algorithm for the uncertainty models with only component $\boldsymbol{Y}$, because the sample of $\boldsymbol{X}$ leads to time-dependent duration which is not considered in deterministic RCPSP.

All algorithms are implemented using JAVA[6], 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. Since the expected makespan defined in Equation (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 (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 used by Kleywegt et al. (2002).

7.2 Examination of Our Algorithms

In this section, we experimentally investigate our algorithms in great detail, including performance of different configurations and sensitivity to different instance parameters. We conduct experiments on small sized instances with $N \in \{10, 20, 30\}$ and uncertainty models with both components $\boldsymbol{X}$ and $\boldsymbol{Y}$. Specifically, $\boldsymbol{X}$ is modeled using the dataset in Table 1, and $\boldsymbol{Y}$ is modeled using the normal distributions $\mathcal{N}(d_i^0, \sigma^2)$. We first examine the impact of different sample sizes in Section 7.2.1, followed by the experiments for analyzing different algorithm configurations in Section 7.2.2. Finally, we examine the impact of problem parameters on our algorithms in Section 7.2.3.

6. For Creemers15, we use the program from http://www.stefancreemers.be/software.php. For BPS, the optimal start-time schedules are obtained using IBM ILOG CP Optimizer 12.8.

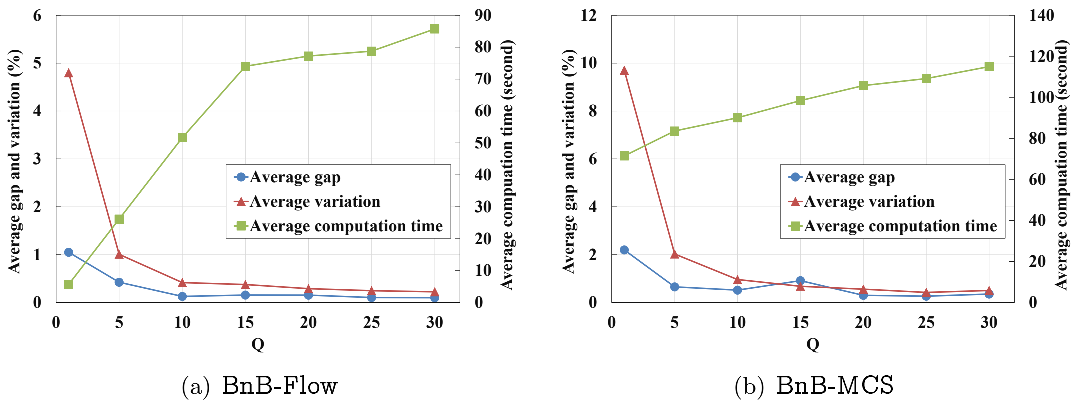


Figure 6: Impact of Sample Size

7.2.1 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 by Kleywegt et al. (2002). To compute the optimality gap ρ, 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:

$$\rho = \left| \hat{g}_{Q_s}(G_R^{rep}) - \frac{1}{Q_{rep}} \sum_{\eta=1}^{Q_{rep}} \hat{g}(G_R^\eta) \right|. \tag{19}$$

Furthermore, we can estimate the variance of ρ as:

$$Var_\rho = \frac{Var_{Q_s}}{Q_s} + \frac{Var_{Q_{rep}}}{Q_{rep}}, \tag{20}$$

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 values 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 6(a) shows the results for `BnB-Flow`, where the average normalized ρ and Var_ρ for an instance subset are plotted, along with the average computation time. Figure 6(b) shows the same curves for `BnB-MCS` on another subset. 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.

Table 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.

7.2.2 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 6.2.4 for `BnB-Flow`. The combination of these heuristics yields four possible configurations of Algorithm 2, 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 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 will use LFT+AEST as the configuration for `BnB-Flow`.

Next, we examine the effectiveness of the constraint propagation (CP) module in `BnB-MCS`. Here we also compare the performance with `BnB-Edge` (Song et al., 2018), which is a POS searching algorithm with the edge-based branching scheme as introduced in Section

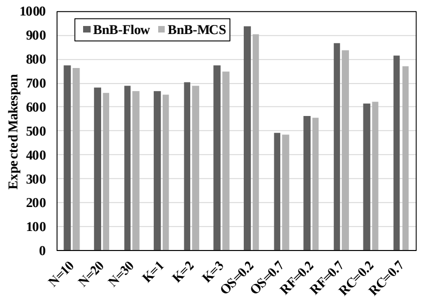
Table 3: Effectiveness of Constraint Propagation

Instance group	BnB-Edge			BnB-Edge+CP		
	Best	Time	PTO	Best	Time	PTO
K=1	636.6	144.58	47.5	632.47	98.15	32.5
K=2	706.9	173.09	56.25	711.82	159.1	52.5
K=3	745.53	206.28	66.25	750.64	192.01	63.75
OS=0.2	928.96	214.18	70.83	931.52	191.08	63.33
OS=0.7	463.73	135.12	42.5	465.1	108.43	35.83
RF=0.2	522.26	114.43	37.5	518.23	63.69	20.83
RF=0.7	870.42	234.87	75.83	878.39	235.82	78.33
RC=0.2	624.55	85.16	25.83	631.97	90.33	30
RC=0.7	768.14	264.13	87.5	764.64	209.18	69.17
Instance group	**BnB-MCS**			**BnB-MCS+CP**		
	Best	Time	PTO	Best	Time	PTO
K=1	617.71	76.54	25	**611.34**	**53.23**	**17.5**
K=2	682.53	94.79	31.25	**662.08**	**81.61**	**25.0**
K=3	715.94	123.99	40	**708.19**	**110.25**	**36.25**
OS=0.2	891.14	145.42	47.5	**868.5**	**124.12**	**40.83**
OS=0.7	452.98	51.46	16.67	**452.57**	**39.28**	**11.67**
RF=0.2	507.62	10.08	25.0	**507.43**	**0.79**	**0**
RF=0.7	836.5	186.8	61.67	**813.65**	**162.6**	**52.5**
RC=0.2	**597.01**	60.2	20	598.37	**48.08**	**15.83**
RC=0.7	747.11	136.68	44.17	**722.7**	**115.32**	**36.67**

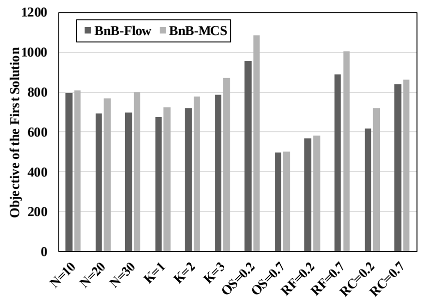
6.3.2. We run the two algorithms with and without constraint propagation on the 240 instances from Set1 with $N = 20$, and summarize the results in Table 3. As shown in this table, constraint propagation can significantly improve the efficiency of the two algorithms. Specifically, for BnB-Edge, the improvements obtained by turning CP on are 14.3% in execution time (174.7 versus 149.8 seconds) and 12.5% in the number of time-out instances (136 versus 119). For BnB-MCS, the algorithm with CP achieves 17% less execution time (98.4 versus 81.7 seconds) and 18.1% less time-out instances (77 versus 63). Moreover, algorithm with the MCS-based branching scheme performs much better than the one with the edge-based branching scheme: with constraint propagation, BnB-MCS is 45.4% faster than BnB-Edge on average, and solves 56 more instances. In the remaining experiments, we will use BnB-MCS with constraint propagation as the POS searching algorithm.

7.2.3 Impact of Problem Parameters

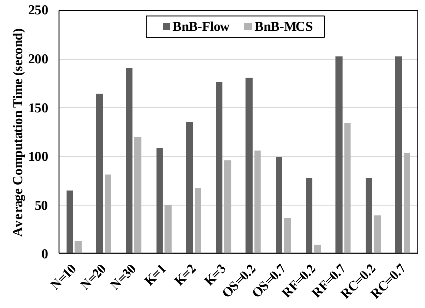
In this section, we examine the efficiency and solution quality of BnB-Flow and BnB-MCS, and analyze the impact of different problem parameters. We use the 720 instances in Set1 for these experiments. In general, BnB-Flow solves 402 (55.8%) instances optimally with an average computation time of 140.1 seconds. In comparison, BnB-MCS solves 559 (77.6%) instances optimally in 71.4 seconds on average. We believe the better efficiency of the MCS-based algorithm is because the search space of POS is smaller that of AON-flow Network, since a POS could accommodate multiple feasible AON-flow Networks, as we have explained in the beginning of Section 6.3. On the other hand, the solution quality given by both



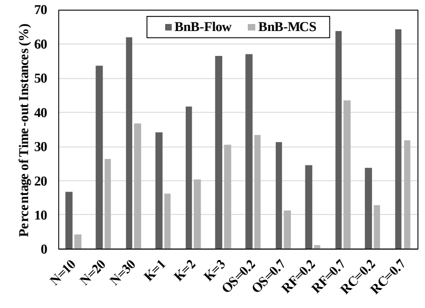
(a) Average Expected Makespan



(b) Average First Objective Value



(c) Average Computation Time



(d) Average Percentage of Time-out Instances

Figure 7: Impact of Problem Parameters on Algorithm Performance

algorithms is rather close (difference is within 3%), as shown in Figure 7(a) where we plot the average expected makespan of the solutions produced by the two algorithms, classified by the problem parameters. To further investigate this observation, we plot the average SAA objective of the first feasible solutions found by the two algorithms in Figure 7(b). As shown, the first solutions returned by BnB-Flow tend to have higher quality than that of BnB-MCS (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 the time aspect. 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. Based on these observations, we believe BnB-Flow can find high-quality solutions even if the search space is not exhausted. In fact, BnB-Flow returns the optimal solutions for 66 instances in the 157 ones that are closed by BnB-MCS but remain open for BnB-Flow.

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 7(c) and 7(d), 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

Table 4: Quality of Solutions on Models with Both Components - Set1

Instance group	BnB-Flow AvgObj	BnB-MCS AvgObj	RLS AvgObj	RLS Diff(%)[1]	ESTA-Iter AvgObj	ESTA-Iter Diff(%)	EBA-Minflow AvgObj	EBA-Minflow Diff(%)
N=10	772.66	**762.16**	792.03	3.92	793.21	4.07	804.31	5.53
N=20	683.39	**660.54**	697.83	5.65	705.18	6.76	743.33	12.53
N=30	689.72	**665.68**	695.12	4.42	709.22	6.54	782.34	17.52
K=1	665.4	**651.03**	674.2	3.56	669.69	2.87	704.75	8.25
K=2	704.9	**689.26**	726.96	5.47	726.91	5.46	767.59	11.36
K=3	775.46	**748.08**	783.83	4.78	811.01	8.41	857.63	14.64
OS=0.2	939.63	**905.92**	958.5	5.8	979.61	8.13	1059.12	16.91
OS=0.7	490.88	**486.33**	498.16	2.43	492.13	1.19	494.19	1.62
RF=0.2	562.43	**554.03**	575.54	3.88	565.6	2.09	564.93	1.97
RF=0.7	868.08	**838.22**	881.12	5.12	906.14	8.1	988.38	17.91
RC=0.2	**615.58**	620.49	637.79	3.61	644.39	4.68	727.3	18.15
RC=0.7	814.93	**771.76**	818.86	6.1	827.34	7.2	826.01	7.03

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

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 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.

7.3 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 listed in Section 7.1. We first conduct a relatively detailed comparison of all algorithms using the small sized instances with $N \in \{10, 20, 30\}$ on the uncertainty models with both components, and with only individual component $\boldsymbol{X}$ and $\boldsymbol{Y}$ in Sections 7.3.1, 7.3.2 and 7.3.3, respectively. Then, we present the results on large sized instances with $N \in \{60, 120\}$ in Section 7.3.4, using uncertainty models with both components.

7.3.1 Results on Models with Both Components

The model $\boldsymbol{U}$ used in this subsection is the same as the one used in Section 7.2. We first report and analyze the results on instances from Set1, which is listed in Table 4. As shown in this table, the results of our two branch-and-bound algorithms are better than all the three benchmark algorithms, and BnB-MCS are the best in most of the instance groups. Meanwhile, BnB-Flow, BnB-MCS and RLS produce better results than the other two general-purpose algorithms, which clearly shows the advantage of incorporating the stochastic knowledge in generating proactive POS. We also observe that EBA-Minflow performs worse than ESTA-Iter. A possible reason is that EBA-Minflow focuses more on the

Table 5: Quality of Solutions on Models with Both Components - Set2

Instance group	BnB-Flow AvgObj	BnB-MCS AvgObj	RLS AvgObj	RLS Diff(%)	ESTA-Iter AvgObj	ESTA-Iter Diff(%)	EBA-Minflow AvgObj	EBA-Minflow Diff(%)
N=10	1009.73	**965.27**	1036.21	7.35	1096.28	13.57	1176.56	21.89
N=20	1044.3	**1014.6**	1068.74	5.34	1168.89	15.21	1354.62	33.51
N=30	1094.51	**1082.56**	1094.76	1.13	1223.02	12.98	1532.78	41.59
K=1	955.17	**934.31**	980.43	4.94	1050.44	12.43	1252.15	34.02
K=2	1047.7	**1021.09**	1066.91	4.49	1166.48	14.24	1352.2	32.43
K=3	1145.68	**1107.03**	1152.37	4.1	1271.26	14.84	1459.61	31.85
OS=0.2	1276.89	**1246.59**	1296.93	4.04	1504.11	20.66	1819.71	45.97
OS=0.5	822.14	**795.02**	836.21	5.18	821.35	3.31	889.59	11.9
RF=0.7	977.05	**956.29**	1005.89	5.19	1068.64	11.75	1222.25	27.81
RF=0.9	1121.98	**1085.32**	1127.25	3.86	1256.82	15.8	1487.06	37.02
RC=0.2	**852.82**	854.07	900.58	5.6	932.21	9.31	1163.81	36.47
RC=0.4	1246.21	**1187.55**	1232.56	3.79	1393.24	17.32	1545.49	30.14

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 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 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 close 133 and 324 instances with the average computation time of 248.1 and 171.5 seconds, respectively. The results are summarized in Table 5, which shows a more prominent improvement against the general-purpose algorithms ESTA-Iter and EBA-Minflow. For RLS which also exploits the stochastic knowledge, the improvement is slightly lower than that in Table 4. We currently do not have an explanation for this observation, and experimental study is needed in the future to better understand the behavior of RLS.

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

In this section, we summarize the experiments on small sized instances and uncertainty models with only component $\boldsymbol{X}$ modeled by the dataset in Table 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 Tables 6 and 7, respectively. Compared to the corresponding values in Tables 4 and 5, the expected makespan values in these two tables are smaller. This is reasonable since now

Table 6: Quality of Solutions on Models with Component $\boldsymbol{X}$ - Set1

Instance group	BnB-Flow AvgObj	BnB-MCS AvgObj	RLS AvgObj	RLS Diff(%)	ESTA-Iter AvgObj	ESTA-Iter Diff(%)	EBA-Minflow AvgObj	EBA-Minflow Diff(%)
N=10	699.36	**697.27**	729.33	4.6	722.31	3.59	744.01	6.7
N=20	627.75	**614.06**	640.92	4.37	654.82	6.64	722.88	17.72
N=30	639.25	**618.23**	643.21	4.04	656.47	6.18	753.06	21.81
K=1	608.62	**595.24**	618.14	3.85	615.6	3.42	673.94	13.22
K=2	646.7	**638.34**	663.63	3.96	667.45	4.56	731.31	14.57
K=3	711.04	**695.99**	731.68	5.13	750.55	7.84	814.69	17.05
OS=0.2	842.39	**821.37**	865.27	5.34	887.06	8	1001.15	21.89
OS=0.7	468.51	**465.01**	477.03	2.58	468.67	0.79	478.81	2.97
RF=0.2	512.16	**503.17**	517.21	2.79	514.44	2.24	520.2	3.38
RF=0.7	798.75	**783.21**	825.09	5.35	841.29	7.41	959.76	22.54
RC=0.2	548.6	**548.27**	565.7	3.18	573.08	4.53	695.12	26.78
RC=0.7	762.31	**738.11**	776.61	5.22	782.65	6.03	784.84	6.33

Table 7: Quality of Solutions on Models with Component $\boldsymbol{X}$ - Set2

Instance group	BnB-Flow AvgObj	BnB-MCS AvgObj	RLS AvgObj	RLS Diff(%)	ESTA-Iter AvgObj	ESTA-Iter Diff(%)	EBA-Minflow AvgObj	EBA-Minflow Diff(%)
N=10	897.02	**878.16**	945.64	7.68	996.22	13.44	1127.67	28.41
N=20	**941.3**	941.66	997.41	5.96	1079.82	14.72	1333.94	41.71
N=30	992.41	**991.35**	1013.05	2.19	1117.48	12.72	1513.08	52.63
K=1	**852.64**	862.31	901.3	5.71	957.02	12.24	1213.79	42.36
K=2	937.84	**931.03**	984.54	5.75	1070.7	15	1340.17	43.95
K=3	1040.23	**1017.82**	1070.27	5.15	1165.79	14.54	1420.72	39.58
OS=0.2	**1119.1**	1130.08	1195.1	6.79	1369.51	22.38	1754.99	56.82
OS=0.5	768.05	**744.03**	775.63	4.25	759.5	2.08	894.8	20.26
RF=0.7	874.67	**870.4**	917.82	5.45	968.65	11.29	1210.94	39.12
RF=0.9	1012.47	**1003.71**	1052.91	4.9	1160.36	15.61	1438.86	43.35
RC=0.2	750.16	**749.57**	797.98	6.46	826.36	10.24	1142.79	52.46
RC=0.4	1136.98	**1124.54**	1172.76	4.29	1302.65	15.84	1507	34.01

only one uncertainty source exists. We also have similar observations as the ones in Section 7.3.1, which can be explained by similar rationale.

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

In this section, we report the experiments on small sized instances and 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 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 (5) is reduced to the traditional stochastic RCPSP. As mentioned in Section 7.1, we use two additional benchmark algorithms in this section, including the state-of-the-art solver Creemers15 and a heuristic BPS. Since Creemers15 gives the optimal expected makespan (w.r.t. elementary policy) when the activity duration follows exponential distribution, i.e. $\boldsymbol{Y}_i \sim Exp(1/d_i^0)$, here we conduct experiments on Set1 and Set2 with exponential distributions, and compare

Table 8: Quality of Solutions on Models with Component $\boldsymbol{Y}$ - Set1

Instance group	BnB-Flow[1]	BnB-MCS	RLS	BPS	ESTA-Iter	EBA-Minflow
N=10	**0.43**	0.47	4.27	2.26	2.3	2.39
N=20	3.29	**2.41**	8.01	4.28	5.79	6.58
N=30	5.15	**4.1**	9.61	5.45	7.74	8.33
K=1	1.62	**1.43**	4.92	3.04	2.75	3.96
K=2	3.12	**2.39**	7.58	3.95	5.47	5.95
K=3	3.81	**2.9**	9.04	4.8	7.24	7.01
OS=0.2	4.75	**4.19**	9.69	4.72	8.49	10.06
OS=0.7	1.13	**0.47**	4.91	3.21	2.12	1.61
RF=0.2	0.75	**0.24**	3.81	2.15	1.73	1.13
RF=0.7	4.92	**4.19**	10.48	5.66	8.51	10.05
RC=0.2	**1.77**	1.87	5.92	4.83	5.4	6.38
RC=0.7	3.95	**2.62**	8.47	3.04	4.94	4.93

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

Table 9: Quality of Solutions on Models with Component $\boldsymbol{Y}$ - Set2

Instance group	BnB-Flow	BnB-MCS	RLS	BPS	ESTA-Iter	EBA-Minflow
N=10	5.78	5.34	10.39	**4.56**	11.88	13.44
N=20	13.01	13.27	20.24	**12.44**	23	25.15
N=30	17.56	17.96	22.62	**15.32**	24.56	23.21
K=1	8.3	9.28	13.79	**8.13**	15	20
K=2	12.43	12.41	17.84	**11.03**	19.99	19.55
K=3	14.7	13.94	20.79	**12.4**	23.62	21.82
OS=0.2	**15.07**	16.04	22.17	15.5	29.66	31.37
OS=0.5	8.92	8.15	13.29	**6.06**	10.46	10.62
RF=0.7	9.67	9.61	15.15	**8.7**	16.42	17.69
RF=0.9	14.02	14.18	19.86	**12.38**	22.73	23.26
RC=0.2	**7.01**	7.29	12.98	9.58	14.54	16.41
RC=0.4	16.76	16.59	22.13	**11.52**	24.7	24.61

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, 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, we give Creemers15 the same computational resources as our algorithms, i.e. the same machine and the same time limit. For Set1 and Set2, Creemers15 solves 690 and 685 instances, respectively. Below we only report the results for the instances solved by Creemers15.

The results are summarized in Tables 8 and 9. For Set1, our two algorithms outperforms all the benchmarks and can find solutions within 5% to the optimal expected makespan, while BnB-MCS tends to perform better than BnB-Flow. For Set2, on one hand, the gaps

Table 10: Quality of Solutions on Models with Both Components - Set1

Instance group	BnB-Flow AvgObj	BnB-MCS AvgObj	RLS AvgObj	RLS Diff(%)	ESTA-Iter AvgObj	ESTA-Iter Diff(%)	EBA-Minflow AvgObj	EBA-Minflow Diff(%)
N=60	715.68	**714.28**	733.86	2.74	730.34	2.25	744.36	4.21
N=120	**991.36**	992.08	1011.14	2	1008.76	1.76	1029.87	3.88
K=1	802.06	**801.4**	817.81	2.05	814.81	1.67	837.78	4.54
K=2	867.54	**866.81**	889.65	2.64	883.23	1.89	901.12	3.96
K=3	**890.96**	891.33	910.04	2.14	910.62	2.21	922.46	3.53
OS=0.2	866.51	**866.33**	902.16	4.14	897.46	3.59	930.89	7.45
OS=0.7	840.54	**840.03**	842.84	0.33	841.64	0.19	843.34	0.39
RF=0.2	731.1	**730.12**	744.56	1.98	732.98	0.39	741.18	1.51
RF=0.7	**975.95**	976.24	1000.44	2.51	1006.12	3.09	1033.06	5.85
RC=0.2	**705.03**	705.07	721.23	2.3	727.83	3.23	762.25	8.12
RC=0.7	1002.02	**1001.29**	1023.77	2.25	1011.27	1	1011.99	1.07

Table 11: Quality of Solutions on Models with Component $\boldsymbol{X}$ - Set2

Instance group	BnB-Flow AvgObj	BnB-MCS AvgObj	RLS AvgObj	RLS Diff(%)	ESTA-Iter AvgObj	ESTA-Iter Diff(%)	EBA-Minflow AvgObj	EBA-Minflow Diff(%)
N=60	**719.32**	721.09	763.09	6.08	822	14.27	849.05	18.03
N=120	**894.16**	894.84	945.04	5.69	990.47	10.77	1019.81	14.05
K=1	**758.71**	760.74	797.01	5.05	837.8	10.42	901.64	18.84
K=2	**808**	809.46	856.09	5.95	911.41	12.8	932.84	15.45
K=3	**853.51**	853.68	909.08	6.51	969.48	13.59	968.8	13.51
OS=0.2	**879.28**	882.08	960.7	9.26	1063.89	21	1103.43	25.49
OS=0.5	734.21	**733.84**	747.42	1.85	748.58	2.01	765.43	4.3
RF=0.7	**768.17**	769.39	812.18	5.73	852.04	10.92	899.15	17.05
RF=0.9	**845.31**	846.54	895.94	5.99	960.43	13.62	969.71	14.72
RC=0.2	670.88	**670.87**	709.77	5.8	733.54	9.34	779.83	16.24
RC=0.4	**942.6**	945.05	998.35	5.91	1078.93	14.46	1089.03	15.53

become larger for all algorithms, which is probably because the parameter configuration for Set2 results in a larger policy space for Creemers15, hence gives more possibility for finding an optimal policy that has a much better expected makespan than the optimal POS. On the other hand, our algorithms BnB-Flow and BnB-MCS are outperformed by BPS on most of the instance groups. This may be caused by the low computational efficiency of our algorithms on this instance set: 610 and 411 instances are time-out for BnB-Flow and BnB-MCS, respectively. This leaves us a direction for future work, which is to further improve the efficiency by exploiting useful properties of the component $\boldsymbol{Y}$.

7.3.4 Results on Large Sized Instances

In this section, we compare our algorithms and the three benchmarks RLS, ESTA-Iter and EBA-Minflow on large sized instances with 60 and 120 activities. We conduct experiments on two sets of instances generated as we described in Section 7.1, each with 480 instances, on the same uncertainty model with both components as the one we used in Section 7.3.1.

Results are summarized in Tables 10 and 11. As shown in these tables, our algorithms still can outperform the other ones, but the improvements are lower than those shown in Tables 4 and 5. This observation is expected, since these instances are much harder to be solved optimally than the small sized ones, due to the NP-hardness of the SAA problem. Within the time limit of 300 seconds, `BnB-Flow` closes 138 and 34 instances in Set1 and Set2 respectively, while `BnB-MCS` can solve 220 and 77 ones optimally in the corresponding sets. Another observation is that in these large sized instances, `BnB-Flow` may perform better than `BnB-MCS` on some instance groups, especially in Set2. This is probably because the CP procedure in `BnB-MCS` is less efficient on these large sized instances.

8. Conclusion and Future Work

Most of the existing works on proactive project scheduling are based on the assumption that the duration uncertainty is not related to the start time of the activity, which may not hold in real-world scenarios. In this paper, we relax this assumption and study the problem of proactive scheduling under a generalized model of uncertain activity durations, which covers both the traditional time-independent uncertainty and the time-dependent workability uncertainty. Based on this model, we formulate the proactive scheduling problem as a stochastic optimization problem, which aims at finding a POS that minimizes the expected makespan. However, this 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. Results show that our algorithms can effectively generate high-quality proactive solutions by exploiting the stochastic knowledge of the uncertainty.

In the future, we plan to improve our current approaches in several ways. Firstly, an immediate direction is to further improve the computational efficiency of our algorithms by introducing other components, such as stronger lower bounds, more effective branching heuristics, and dominance rules. Secondly, we aim at study how to apply our current algorithms on the general time-dependent uncertainty models. Finally, we intend to extend the current approach to handle RCPSP with minimum and maximum time lags (RCPSP/max) (Fu et al., 2012), which could further improve the generality of our approach.

Acknowledgments

This work was conducted within the Rolls-Royce@NTU Corporate Lab with support from the National Research Foundation (NRF) Singapore under the CorpLab@University Scheme. Zhiguang Cao (corresponding author) was partially supported by National Natural Science Foundation of China (61803104). The authors wish to thank all reviewers for their insightful comments.

References

Artigues, C., Michelon, P., & 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., & 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.

Aytug, H., Lawley, M. A., McKay, K., Mohan, S., & Uzsoy, R. (2005). Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, *161*(1), 86–110.

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

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

Bidot, J., Vidal, T., Laborie, P., & 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., & Kan, A. R. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, *5*(1), 11–24.

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

Cao, Z., Guo, H., Zhang, J., Niyato, D., & Fastenrath, U. (2016). Finding the shortest path in stochastic vehicle routing: A cardinality minimization approach. *IEEE Transactions on Intelligent Transportation Systems*, *17*(6), 1688–1702.

Cao, Z., Guo, H., Zhang, J., Oliehoek, F. A., & Fastenrath, U. (2017). Maximizing the probability of arriving on time: A practical q-learning method.. In *Proceedings of the 31th AAAI Conference on Artificial Intelligence (AAAI)*, pp. 4481–4487.

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., & Williams, B. C. (2015). Optimising bounds in simple temporal networks with uncertainty under dynamic controllability constraints.. In *ICAPS*, pp. 52–60.

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

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

Fu, N., Lau, H. C., & 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., & 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., & Lau, H. C. (2016). Robust partial order schedules for rcpsp/max with durational uncertainty. In *ICAPS*, pp. 124–130. AAAI Press.

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

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

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

Kleywegt, A. J., Shapiro, A., & 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.

Laborie, P. (2005). Complete mcsv-based search: Application to resource constrained project scheduling. In *IJCAI*, pp. 181–186.

Lamas, P., & 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., & Herroelen, W. (2011). Time slack-based techniques for robust project scheduling subject to resource uncertainty. *Annals of Operations Research*, *186*(1), 443–464.

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

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

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

Morris, P., & Muscettola, N. (2005). Temporal dynamic controllability revisited. In *AAAI*, pp. 1193–1198.

Morris, P., Muscettola, N., & Vidal, T. (2001). Dynamic control of plans with temporal uncertainty. In *IJCAI*, pp. 494–499.

Muscettola, N. (2002). Computing the envelope for stepwise-constant resource allocations. In *International Conference on Principles and Practice of Constraint Programming*, pp. 139–154. Springer.

Patterson, J. H., Talbot, F. B., Slowinski, R., & Węglarz, J. (1990). Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems. *European Journal of Operational Research*, *49*(1), 68–79.

Policella, N., Cesta, A., Oddi, A., & Smith, S. F. (2007). From precedence constraint posting to partial order schedules. *Ai Communications*, *20*(3), 163–180.

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

Policella, N., Smith, S. F., Cesta, A., & Oddi, A. (2004). Generating robust schedules through temporal flexibility.. In *ICAPS*, Vol. 4, pp. 209–218.

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

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

Song, W., Kang, D., Zhang, J., & 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)*, pp. 6278–6285.

Sprecher, A., & Drexl, A. (1998). Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm1. *European Journal of Operational Research*, *107*(2), 431–450.

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

Vanhoucke, M., Coelho, J., Debels, D., Maenhout, B., & 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., & Lau, H. C. (2016). A proactive sampling approach to project scheduling under uncertainty. In *AAAI*.