Proactive Project Scheduling with Time-dependent Workability Uncertainty

Wen Song, Donghun Kang, Jie Zhang[a], Hui Xi[b]
Rolls-Royce@NTU Corp Lab, Nanyang Technological University, Singapore
[a]School of Computer Science and Engineering, Nanyang Technological University, Singapore
[b]Rolls-Royce Singapore Pte Ltd, Singapore
{songwen, donghun.kang, [a]zhangj}@ntu.edu.sg, [b]hui.xi@rolls-royce.com

ABSTRACT

Proactive scheduling can effectively handle activity duration uncertainty in real-world projects, by generating a baseline solution according to a prior stochastic knowledge. However, most of the previous approaches cannot deal with the activity duration uncertainty caused by time-dependent workability uncertainty. In this paper, we aim at finding a partial-order schedule (POS) that produces the minimum expected makespan on a given probability model of workability uncertainty. Since this is a hard discrete stochastic optimization problem, we propose an approximation approach based on Sample Average Approximation (SAA), and develop a branch-and-bound algorithm to optimally solve the SAA problem. Empirical results on benchmark problem instances and real-world distribution data show that our approach outperforms the best general-purpose POS generation approaches that do not exploit the stochastic knowledge.

CCS Concepts

•Computing methodologies → Planning under uncertainty;

Keywords

Proactive project scheduling, time-dependent duration uncertainty, sample average approximation, resource allocation

1. INTRODUCTION

Project managers in the industry often need to deal with Resource-Constrained Project Scheduling Problem (RCPSP), a general model for describing and solving various scheduling problems, to arrange, control and optimize their business processes. While deterministic RCPSP and its extensions are well studied for decades (e.g. [10, 17, 28]), the assumptions of perfect information and deterministic problem parameters are frequently violated in practice. Since real-world project activities are usually highly sensitive to different uncertainties (e.g. transportation time, manpower availability, weather changes), models and approaches incorporating uncertainties are of great practical value. Proactive scheduling refers to a class of approaches that exploit a prior knowledge about the uncertainties to generate a baseline solution, either a schedule or policy, before execution. Compared to complete online scheduling (e.g. [22, 29]) that generates no baseline solution and rescheduling (e.g. [2, 11]) that continuously revises the baseline solution, proactive approaches tend to produce solutions with higher quality and robustness [5]. Moreover, decisions made in the baseline solutions (e.g. activity start times, resource allocation commitments) could serve important functionalities in providing visibility for better coordination of the execution process [20, 23].

A number of proactive approaches have been developed for project scheduling under uncertainty [13, 14]. Most of them deal with models where the duration of each activity is modeled as a random variable that is independent of its scheduled start time [4]. However, as mentioned in [7], this model cannot cover the full spectrum of uncertainty sources. For example, in a quality assurance project, each testing activity must secure enough times or days to carry out the test successfully under certain weather conditions (e.g. temperature, humidity, wind speed). In other words, the uncertainty on the activity duration comes from the *workability* of each time slot in the scheduling horizon. Due to seasonality, duration of an activity could be affected differently by its start time (e.g. July or December). As will be mentioned in Section 2, previous approaches cannot handle this case.

This paper studies proactive scheduling for RCPSP under time-dependent workability uncertainty, with the objective of minimizing the expected makespan. We present a reasoning approach for an autonomous agent responsible for proactive scheduling on behalf of the project manager. Instead of generating a complete schedule, our agent computes a partial-order schedule (POS) [27] as a solution, since it is more flexible in handling unforeseen events [13]. Several approaches in [3, 26, 27] can generate a feasible POS for a given RCPSP instance. However, they are general-purpose approaches and do not use a prior stochastic knowledge. In contrast, we propose a stochastic optimization model that explicitly incorporates the workability uncertainty model.

It is non-trivial for our agent to find the optimal solution; RCPSP is known to be intractable [6], and the presence of uncertainty makes it more complex [4]. To address these challenges, we propose a novel approach based on Sample Average Approximation (SAA) [15], which provides a general framework for approximately solving stochastic optimization problems, with the proven ability of converging to the optimal solution. Our approach first generates a set of samples from the workability uncertainty model, then optimally solves the SAA problem built on these samples through a branch-and-bound process. By exploiting the

Appears in: *Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*

properties of the SAA problem, we design several techniques for pruning, branching and lower bounding. Empirical results on benchmark problems and real-world distribution data show that our approach can generate solutions with higher quality than the best general-purpose approaches.

2. RELATED WORK

In this section, we briefly review the works in proactive project scheduling. Based on the taxonomy in [5], we classify existing approaches into two categories according to their solution types. The first category of approaches generates a generic solution that makes complete decisions, such as a start-time schedule. These approaches often try to optimize a risk-aware objective, i.e. the schedule should be valid with a high probability in execution time. The current best approaches [18, 31] in this category introduce a probability constraint to the deterministic problem model to restrict the probability of schedule invalidation. Both approaches rely on SAA to handle the hardness of the probability constraint. However, their models and approaches cannot be applied to solve our problem, since the samples of activity duration cannot be obtained from the workability distributions without knowing their start times.

Different from those in the first category, approaches in the second category generate flexible solutions that make part of the decisions, and leave the remaining ones to the execution time. A typical flexible solution is the dynamically controllable Simple Temporal Network with Uncertainty (STNU) [8, 25, 24]. However, STNU based approaches mainly focus on temporal reasoning. They are not directly applicable to the context of project scheduling since they cannot handle resource constraints. Another way to generate flexible solutions is the redundancy-based techniques [9, 19], which insert extra slacks to protect activities against machine breakdowns. Their problem settings are somehow similar to ours, since the machine breakdown probability is related to the time slot in the horizon. However, their approaches are limited to specific probability distributions (e.g. normal [9] and exponential [19]) of machine breakdown. In contrast, our approach does not put any assumption on the stochastic knowledge about the workability uncertainty.

POS is another type of flexible solutions. A POS specifies an additional set of precedence constraints, such that any temporal feasible schedule is also resource feasible. In [27], two approaches that directly generate POS from RCPSP instances are proposed, namely 1) Envelop Based Algorithm (EBA) that generates a POS by detecting and removing potential resource conflicts and 2) Earliest Start Time Algorithm (ESTA$^{\mathrm{C}}$) that transforms an earliest start time solution to a POS using a chaining process. For EBA, the current best resource conflict detection algorithm is the min-flow approach [21], while the state-of-the-art ESTA$^{\mathrm{C}}$ algorithm is the iterative chaining procedure [26]. A major limitation of these approaches is that they do not exploit the stochastic knowledge about uncertainties. As will be shown in our experiments, significant improvement can be obtained by our approach, where the stochastic knowledge is explicitly considered in generating POS. Recently, several approaches are developed to generate a robust POS that is adapted to the known activity duration distributions [4, 12, 13]. However, a major assumption of these approaches is that the probability model of activity durations is independent of activity start times, which cannot hold in our problem.

3. PRELIMINARIES

We first introduce the basic notations and concepts.

3.1 Deterministic RCPSP

In a deterministic RCPSP instance, a set of non-preemptive activities $A = \{a_1, ..., a_N\}$ from a project needs to be scheduled on a set of renewable resources $R = \{r_1, ..., r_K\}$, with a horizon of T consecutive time slots. Each $a_i \in A$ has a fixed duration of $d_i^s \in \mathbb{N}$ time slots, and requires $b_{ik} \in \mathbb{N}$ units of resource $r_k \in R$, which has a finite capacity of $c_k \in \mathbb{N}$ units in each time slot. Usually two dummy activities a_0 and a_{N+1} having zero durations and no resource requirement are added to represent the start and completion of the project. Denote $A_p = A \cup \{a_0, a_{N+1}\}$. A pair of activities in A_p could have a precedence relation $a_i \prec a_j$, indicating that a_j must start after the completion of a_i. Denote the set of all precedence relations as $E = \{(i,j) | a_i \prec a_j, \forall i, j \in \{0, ..., N+1\}\}$. The precedence relations within a project can be represented as an Activity-On-Node (AON) network, which is a graph $G = (A_p, E)$. Throughout this paper, we denote $V(G)$ and $E(G)$ as vertex and edge sets of a graph G, respectively. We also denote $Tr(G)$ as the transitive closure of G, where $(i,j) \in Tr(G)$ indicates there is a path from a_i to a_j.

A schedule of RCPSP is a vector $S = (s_0, ..., s_{N+1})$, where s_i is the start time of a_i. For the deterministic RCPSP, once s_i is determined, the completion time $c_i = s_i + d_i^s$ of a_i 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 $MS(S) = \max_i\{c_i\}$.

3.2 Partial-order Schedule

A POS is a graph $G_R = (A_p, E \cup E_R)$ that augments the AON network by adding an additional set of precedence constraints E_R, such that any temporal feasible solution of G_R is also resource feasible [27]. The advantage of POS in handling uncertain activity durations is that it provides an efficient way to obtain a feasible solution by simply propagating the realized activity durations d_i through the POS, without the need of complex reasoning on the resource capacity constraints. Suppose a POS G_R is generated as a proactive solution. During execution time, the start time s_i of a_i can be computed very easily using the equation below:

$$s_i = \max\{c_j = s_j + d_j | (j,i) \in E \cup E_R\}. \quad (1)$$

In other words, a_i is started after the completion of all its predecessors in G_R. Along with execution, the start times of all $a_i \in A_p$ can be calculated, hence a schedule is obtained.

One limitation of the POS defined above is that it does not include the resource allocation decisions (i.e. which resource unit is allocated to which activity), which are considered to be important in preparing and coordinating the execution process [20]. Therefore, in this paper we use the AON-flow Network [3] as our solution. Essentially, AON-flow Network can be considered as a special type of POS where the resource allocation decisions are explicitly specified by a set of resource flows between activities. Denote an AON-flow Network as $G_F = (A_p, E \cup E_F)$. Each edge $(i,j) \in E_F$ is associated with a resource flow 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. It should be noted that $E \cap E_F$ is not necessarily to be $\emptyset$, i.e. some edges in E may also carry resource flows. An AON-flow Network is feasible if it

satisfies the following conditions:

$$\begin{aligned}
&\exists r_k \in R, f_{ijk} > 0, && \forall (i,j) \in E_F, \\
&\sum_{(j,i)\in E_F} f_{jik} = b_{ik} = \sum_{(i,j)\in E_F} f_{ijk}, && \forall i \in A, r_k \in R, \\
&\sum_{(0,j)\in E_F} f_{0jk} = c_k, && \forall r_k \in R, \\
&\sum_{(j,N+1)\in E_F} f_{j,(N+1),k} = c_k, && \forall r_k \in R,
\end{aligned} \tag{2}$$

where the first condition guarantees that an edge in E_F indeed carries resource flows, and the remaining three guarantee that the resource flows are balanced. We denote the set of all feasible AON-flow Networks that satisfy the four conditions in Equation (2) as $\boldsymbol{G}_F$.

4. PROBLEM FORMULATION

In this section, we formulate our proactive scheduling problem. We first introduce the workability uncertainty. Without loss of generality, we assume all activities in A can be classified into Z types, and activities of the same type have the same workability uncertainty (e.g. requiring the same weather condition). Denote the type of a_i as $z_i \in \{1, ..., Z\}$. For each type z, we model the activity workability using a random vector $X_z = (X_{z1}, ..., X_{zT})$, where X_{zt} is a random variable representing the workability of a time slot $t \in \{1, ..., T\}$ for type z. Let x_{zt} be the realization of X_{zt}. Here we assume $x_{zt} \in \{0, 1\}$, where activities of type z can only work on t when $x_{zt} = 1$.[1] We call the collection $X = (X_1, ..., X_Z)$ the workability uncertainty model.

Under the workability uncertainty, the duration of a_i is no longer a deterministic value d_i^s, but a random variable D_i. Meanwhile, d_i^s is a condition for determining the completion of a_i: an activity must acquire at least d_i^s workable time slots before completion. Denote the realization of D_i as d_i, the cumulative distribution function of D_i conditioning on s_i can be formulated as:

$$\begin{aligned}
F_{D_i}(d_i \mid s_i = t) &= P(D_i \le d_i \mid s_i = t) \\
&= P\left(\sum_{\tau=t}^{t+d_i-1} X_{z_i\tau} \ge d_i^s \,\middle|\, s_i = t \right).
\end{aligned} \tag{3}$$

Depending on application, it may be costly or inconvenient to obtain the probability distribution in Equation (3). For example, X_z may be modeled using a stochastic process, or even implicitly embedded in a simulator. Moreover, the dependence of start time makes it even harder for scheduling, since the duration of an activity depends on the scheduling decisions of other activities. Therefore, we directly exploit the workability model, and formulate the problem studied in this paper as follows: given a RCPSP instance and the workability uncertainty model X, find a feasible AON-flow Network $G_F^* \in \boldsymbol{G}_F$ that minimizes the expected makespan:

$$G_F^* = \underset{G_F \in \boldsymbol{G}_F}{\operatorname{argmin}} \{ g(G_F) = \mathbb{E}[MS(G_F, X)] \}, \tag{4}$$

where $MS(G_F, X)$ is a random variable representing the (stochastic) makespan of G_F on X.

[1]This 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.

5. SAMPLE AVERAGE APPROXIMATION

Equation (4) describes a hard stochastic optimization problem, not only due to the combinatorial nature of RCPSP. In fact, given G_F, it is costly to even compute the exact expected value $g(G_F)$. This is because the number of possible realizations of X is 2^{ZT}, which grows exponentially with the problem size. We tackle this hardness using Sample Average Approximation (SAA). The basic idea of SAA is to substitute the original problem with an approximated one, where the original distribution and the expected value function are replaced by a set of independent samples and a sample average function, respectively. To approximate our problem in Equation (4) using SAA, we first denote $x = (x_1, ..., x_Z)$ as a sample generated from the workability uncertainty model X, where $x_z = (x_{z1}, ..., x_{zT})$ is a realization of X_z. Let $\boldsymbol{x} = (x^1, ..., x^M)$ be a set of M randomly generated samples. Then, the SAA problem can be formulated as:

$$\hat{G}_F^* = \underset{G_F \in \boldsymbol{G}_F}{\operatorname{argmin}} \left\{ \hat{g}(G_F) = \frac{1}{M} \sum_{\xi=1}^{M} MS(G_F, x^\xi) \right\}, \tag{5}$$

where $\hat{G}_F^*$ is the optimal solution of the SAA problem, $\hat{g}(G_F)$ is the sample average function of the original expected value function $g(G_F)$ in Equation (4), and $MS(G_F, x^\xi)$ is the makespan of a solution G_F on a sample x^ξ. As proved in [15], with the increase of sample size M, $\hat{G}_F^*$ will converge to G_F^* at an exponential rate.

Next, we discuss the computation of $MS(G_F, x)$. Given a POS G_F, we know that the start time s_i of a_i can be computed using Equation (1), hence we only need to determine the duration d_j on the sample x^ξ of all its predecessors specified by G_F. Given a start time s_i of a_i, a duration d_i is said to be feasible on x if it satisfies the completion condition, i.e. $\sum_{\tau=s_i}^{c_i-1} x_{z_i\tau} \ge d_i^s$, where $c_i = s_i + d_i$ is the completion time. There are many d_i that can satisfy the above condition, but we can show that it is sufficient to use the equation below:

$$d_i(s_i, x) = \min \left\{ d \ge 0 \,\middle|\, \sum_{\tau=s_i}^{s_i+d-1} x_{z_i\tau} \ge d_i^s \right\}. \tag{6}$$

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

LEMMA 1. *Given two start times s_i^1 and s_i^2 of an activity a_i on a sample x, if $s_i^1 \le s_i^2$, then for any feasible duration d_i^2 of s_i^2, $s_i^1 + d_i^1(s_i^1, x) \le s_i^2 + d_i^2$.*

PROOF. We only need to show that $s_i^1 + d_i^1(s_i^1, x) \le s_i^2 + d_i^2(s_i^2, x)$, since $d_i^2(s_i^2, x)$ is less than any other feasible d_i^2. Denote the completion time as $c_i^1 = s_i^1 + d_i^1(s_i^1, x)$, $c_i^2 = s_i^2 + d_i^2(s_i^2, x)$, respectively. According to Equation (6),

$$\sum_{t=s_i^1}^{c_i^1-1} x_{z_i t} = \sum_{t=s_i^2}^{c_i^2-1} x_{z_i t} = d_i^s. \tag{7}$$

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

$$\sum_{t=s_i^1}^{s_i^2-1} x_{z_i t} + \sum_{t=s_i^2}^{c_i^2-1} x_{z_i t} + \sum_{t=c_i^2}^{c_i^1-1} x_{z_i t} = d_i^s. \tag{8}$$

Since the second term in the left hand side of Equation (8) equals to d_i^s, we have $\sum_{t=s_i^1}^{s_i^2-1} x_{z_i t} + \sum_{t=c_i^2}^{c_i^1-1} x_{z_i t} = 0$, indicating $\sum_{t=c_i^2}^{c_i^1-1} x_{z_i t} = 0$. Hence, the third term in the left hand side of Equation (8) can be removed, indicating $d_i' = c_i^2 - s_i^1$ is a feasible duration. However, based on the assumption, $d_i' < c_i^1 - s_i^1 = d_i^1(s_i^1, x)$, which contradicts Equation (6). □

Based on Lemma 1, an activity can never complete earlier on a sample by starting later (though a smaller duration may be obtained). Hence, we can prove the following proposition:

PROPOSITION 1. *Given a partial-order schedule G_F and a sample x, the schedule $S(G_F, x)$ generated by using Equation (6) produces the lowest makespan.*

PROOF. Let $S'(G_F, x)$ be a schedule obtained by setting the duration of an activity a_i to $d_i' > d_i(s_i, x)$. Then, all its immediate successors j can only start no earlier than its start time in $S(G_F, x)$, indicating a completion time equal or larger than that of $S(G_F, x)$ according to Lemma 1. This delay will be further propagated through G_F, and finally lead to a makespan equal or larger than $MS(S(G_F, x))$. □

Let $MS(G_F, x^\xi) = MS(S(G_F, x^\xi))$ be the makespan of G_F on x. Then, we can have another conclusion as below:

OBSERVATION 1. *Given two partial-order schedules G_F^1 and G_F^2, if $V(G_F^1) = V(G_F^2) = A$ and $E(G_F^1) \subseteq E(G_F^2)$, then $MS(G_F^1, x) \leq MS(G_F^2, x)$ holds for any sample x.*

The reason is that, for any $a_i \in A$, it cannot start earlier in $S(G_F^2, x)$ than in $S(G_F^1, x)$, since it needs to respect more precedence constraints specified in $E(G_F^2) \setminus E(G_F^1)$. According to Lemma 1, it cannot complete earlier either. Thus, G_F^2 results in an equal or larger makespan than G_F^1.

The computation of $MS(G_F, x^\xi)$ is very efficient with a complexity of $\mathcal{O}(N^2T)$. Therefore, it is tractable to evaluate the objective $\hat{g}(G_F)$, with a complexity of $\mathcal{O}(MN^2T)$. However, the SAA problem is intractable, as stated below:

PROPOSITION 2. *The SAA problem is NP-hard.*

PROOF. We follow the proof for deterministic RCPSP [6], where it is reduced from a NP-complete problem Partition Into Triangles (PIT): for a graph $G = (V, E)$ where $|V| = 3q$, is there a partition of G into q disjoint subsets, such that each subset contains three pairwise adjacent vertices?

For any PIT instance, we first construct a RCPSP instance as in [6]. For each $i \in V$, an activity a_i is created with $d_i^s = 1$. For each pair $(i, j) \notin E$, a resource r_{ij} with capacity $c_{ij} = 1$ is added, which is only required by a_i and a_j with $b_{i,ij} = b_{j,ij} = 1$, and $b_{l,ij} = 0$ for other activities a_l. Then we construct an instance for the SAA problem, by adding one sample x where $x_{zt} = 1$ for all z and t. We claim that the SAA problem has a solution G_F with $\hat{g}(G_F) \leq t$ if and only if the PIT instance has a solution.

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

6. THE BRANCH-AND-BOUND APPROACH

Due to the intractability of the SAA problem, in this section we design a branch-and-bound approach for efficiently finding the optimal solution to the SAA problem.

6.1 Branching Scheme

Our approach employs a depth-first tree search algorithm to systematically explore the feasible solution domain $\boldsymbol{G}_F$ by continuously extending a partial solution G_F'. To determine the edges (i, j) and the corresponding resource flows f_{ij} in an AON-flow Network, our approach employs a two-level branching scheme, as shown in Algorithm 1. At the first level (activity level), a set of precedence feasible activities ES is identified first, where $ES = \{a_i \in A_p | a_i \notin V(G_F'), a_j \in V(G_F'), \forall (j, i) \in E\}$. If ES is empty, then a feasible solution is reached and the algorithm backtracks (Lines 3-7). Otherwise, an activity a_l is chosen and removed from ES for branching, according to some heuristics (see Section 6.4). Then, a lower bound $LB(G_F', a_l)$ of linking a_l to G_F' is computed to determine whether the search path should be pruned or not. If not, the algorithm enters the second level (link level), where a set of feasible links LK is generated as branching candidates for linking a_l to G_F'. A link $lk = \{(i, l) | i \in V(lk)\}$ is a set of edges that link a set of vertices $V(lk) \subseteq V(G_F')$ in G_F' to a_l. The method of identifying LK will be discussed in Section 6.2. Then, the algorithm iteratively chooses and removes one link lk from LK to branch by calling the function ChooseLink, until LK is empty (Lines 15-20). The function LinkActivity in Line 17 is called to link a_l to G_F' using lk, by generating a new partial solution $\bar{G}_F' = (V(G_F') \cup \{a_l\}, E(G_F') \cup lk)$. The function RemoveActivity in Line 19 does the inverse operation as LinkActivity upon backtracking to remove a_l and lk from G_F'. The branching and pruning at the link level are embedded in ChooseLink (Algorithm 2), and will be discussed in Section 6.2.

During the searching process, an outgoing capacity matrix $OC = [oc_{ik}]_{(N+2)\times K}$ is maintained to record the remaining capacity of resources that can be transferred from a linked activity in $V(G_F')$ to an unlinked one. In the initial OC, denoted as OC^0, oc_{ik}^0 is set to b_{ik} for all $1 \leq i \leq N$, while oc_{0k}^0 and $oc_{N+1,k}^0$ are set to c_k and 0, respectively. In Algorithm 1, when a_l is linked by LinkActivity, oc_{ik} will be set to $oc_{ik} - f_{ilk}$ if lk contains a resource flow from a_i to a_l, and an inverse operation will be conducted in RemoveActivity when backtracking. At the beginning, an initial partial solution $G_F'^0 = (\{0\}, \emptyset)$ is created with the dummy start activity only. Then the searching process is invoked by calling $\text{BnB}(G_F'^0, null, L, OC^0)$, where L is a large double value. If the lower bounds adopted in the algorithm are admissible, i.e. they never overestimate the best objective that can be achieved by the subtree rooted from the corresponding search node, our algorithm guarantees to terminate with the optimal solution to the SAA problem.

6.2 Finding and Choosing Feasible Links

This section describes how feasible links are identified and chosen. We first define the link feasibility. Suppose a partial solution $\bar{G}_F'$ is obtained by linking a_l to G_F' using lk. Then lk is feasible when the following conditions are satisfied:

$$\begin{aligned} &(i, l) \in Tr(\bar{G}_F'), \quad \forall i \in \{i | (i, l) \in E\}, \\ &\sum_{(i,l)\in lk} f_{ilk} = b_{ik}, \quad \forall r_k \in R, \end{aligned} \tag{9}$$

Algorithm 1: BnB($G'_F, \hat{G}^*_F, \hat{g}^*, OC$)

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

```
ES ← FindEligibleActivities(V(G'_F)) ;
if ES = ∅ then
    ĝ' ←ComputeObj(G'_F) ;
    if ĝ' < ĝ* then
        ĝ* ← ĝ';
        Ĝ*_F = G'_F;
    return;
while ES ≠ ∅ do
    a_l ← ChooseActivity(ES);
    ES ← ES \ {a_l};
    LB(G'_F, a_l) ← ComputeLB_A(G'_F, a_l);
    if LB(G'_F, a_l) < ĝ* then
        LK ← FindFeasibleLinks(G'_F, a_l, OC);
        lk ←ChooseLink(G'_F, LK, ĝ*);
        while lk ≠ null do
            LK ← LK \ {lk};
            Ḡ'_F ←LinkActivity(a_l, G'_F, lk, OC);
            BnB(Ḡ'_F, Ĝ*_F, ĝ*, OC);
            G'_F ←RemoveActivity(a_l, Ḡ'_F, lk, OC);
            lk ←ChooseLink(G'_F, LK, ĝ*);
return;
```

where the first one guarantees the precedence constraints regarding a_l in E is respected, and the second one makes sure a_l receives enough resource units from the edges in lk. Next, we show that for our problem, the search space can be limited to integer resource flows.

PROPOSITION 3. *For all k and i, if $c_k \in \mathbb{N}$ and $b_{ik} \in \mathbb{N}$, then it is sufficient to consider only integer flows.*

PROOF. For any AON-flow Network G_F, we first construct a flow network $G^T_F(k)$ for each r_k by augmenting G_F as follows: 1) $\forall a_i, i \neq N+1$, add an edge $(i, N+1)$; 2) set the demands of a_0 and a_{N+1} to c_k and $-c_k$, respectively; 3) split each non-dummy activity a_i into two vertices a_{i_s} and a_{i_t} with demands b_{ik} and $-b_{ik}$, respectively, linked with an edge (i_s, i_t); 4) set the constraint of flow $f_{i_s i_t k}$ of each (i_s, i_t) to $b_{ik} \leq f_{i_s i_t k} \leq b_{ik}$, while other flows with only nonnegative constraints. It is easy to verify that if $G^T_F(k)$ has a feasible flow, then G_F also has a feasible flow where $f_{ijk} = f_{i_t j_s k}, \forall (i,j) \in E_F$. Next, we augment $G^T_F(k)$ to construct another network $G^{T'}_F(k)$: 1) add two virtual vertices, source a_s and sink a_t; 2) for each a_{i_s}, add an edge (s, i_s) with capacity b_{ik}; 3) for each a_{i_t}, add an edge (i_t, t) with capacity b_{ik}; 4) add two edges, $(s, 0)$ with capacity c_k and $(N+1, t)$ with capacity c_k. According to network optimization theory [1], a feasible flow exists in $G^T_F(k)$ if and only if the maximum flow from a_s to a_t in $G^{T'}_F(k)$ saturates all edges added from $G^T_F(k)$. In addition, due to the integrality property [1], if all the edge capacities in $G^{T'}_F(k)$ are integers, then the maximum flow is also integer. Therefore, we can obtain a feasible integer flow in G_F if it is feasible. □

To find all the feasible integer flows, we first identify all $a_i \in G'_F$ with positive oc_{ik} as candidates for linking a_l. Then, all the feasible resource flows are enumerated to form the set LK. Finally, each link in LK is verified against the first condition in Equation (9). If any immediate predecessor a_i of a_l cannot reach a_l by lk, an edge (i, l) with zero resource flow is added to lk to make it precedence feasible[2].

Algorithm 2: ChooseLink($G'_F, LK, \hat{g}^*$)

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

Output: lk: the chosen edge

```
while LK ≠ ∅ do
    lk ←GetLink(LK);
    LB(G'_F, lk) ←ComputeLB_L(G'_F, lk);
    if LB(G'_F, lk) < ĝ* then
        return lk;
    else
        LK ←RemoveLinks(LK, lk);
return null;
```

Due to the combinatorial nature, LK could contain many branching alternatives. To further reduce the size of LK, we design the ChooseLink function that embeds an additional pruning step. As shown in Algorithm 2, ChooseLink tries to find a feasible link $lk \in LK$ that is not pruned by $\hat{g}^*$. It first selects a link lk from LK using some heuristics (see Section 6.4), and then computes its lower bound. If lk is not pruned, the function returns it as the chosen link. Otherwise, a function RemoveLinks is called to remove any lk' satisfying $V(lk) \subseteq V(lk')$. This is based on the following observation:

OBSERVATION 2. *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 2 will be justified in Section 6.3 when the function ComputeLB_L is discussed.[3] Basically, when a link lk is pruned, all links that involve additional activities in the partial solution can also be safely pruned, since they result in equal or larger lower bounds than that of lk.

6.3 Lower Bounds

To guarantee the optimality, two admissible lower bounding functions are needed, i.e. ComputeLB_A for the activity level, and ComputeLB_L for the link level. Before going into details, we first give a general lower bound on the objective function. Given a partial solution G'_F, it is easy to verify that $\hat{g}_{LB}$ in the below equation is a lower bound of $\hat{g}$:

$$\hat{g}_{LB}(G'_F) = \frac{1}{M} \sum_{\xi=1}^{M} MS_{LB}(G'_F, x^\xi), \tag{10}$$

where $MS_{LB}(G'_F, x^\xi)$ is a lower bound of the makespan of G'_F on x^ξ. In other words, to lower bound G'_F on $\hat{g}$, we only need to lower bound G'_F on each individual sample x^ξ. Based on Equation (10), we construct the two lower bounds based on the critical path lower bound for solving the deterministic RCPSP [10]. Basically, for the activities that have not been linked to G'_F, only precedence constraints are considered in computing the makespan. Below we first discuss the lower bounding computation at the link level.

ComputeLB_L. To compute the lower bound of a feasible link lk, we first construct the new partial solution $\bar{G}'_F$ obtained by linking a_l to G'_F using lk. Then, we construct

[2]This does not violate the first condition in Equation (2), since (i, l) simply represents a precedence constraint in E.

[3]Note that not all admissible lower bounds satisfy this observation, but the one we design in Section 6.3 does.

another graph $\bar{G}''_F = (A_p, E(\bar{G}'_F) \cup E)$ that augments $\bar{G}'_F$ by linking $a_i \notin V(\bar{G}'_F)$ to $\bar{G}'_F$ using the edges in E. By propagating $\bar{G}''_F$ on x, we can obtain a temporal feasible schedule $S(\bar{G}''_F, x)$, where $a_i \notin \bar{G}'_F$ is not necessarily resource feasible. According to Observation 1, $MS(\bar{G}''_F, x)$ is an admissible lower bound for the branching choice of linking a_l to G'_F using lk. The reason is that, $\forall G_F \in \boldsymbol{G}_F$ obtained by extending G'_F, we have $E(\bar{G}''_F) \subseteq E(G_F)$ since additional edges are added to resolve resource conflicts. Therefore, by propagating $\bar{G}''_F$ on each sample x^ξ, we have

$$LB(G'_F, lk) = \frac{1}{M} \sum_{\xi=1}^{M} MS(\bar{G}''_F, x^\xi). \tag{11}$$

Now we can show the correctness of Observation 2. For two links lk^1 and lk^2 satisfying the conditions in Observation 2, we have $E(\bar{G}''^1_F) \subseteq E(\bar{G}''^2_F)$ by the above construction process. Based on Observation 1, $LB(G'_F, lk^1) \leq LB(G'_F, lk^2)$.

ComputeLB_A. Unlike in ComputeLB_L, we cannot construct a common solution to propagate on all samples to compute $LB(G'_F, a_l)$, since the feasible links have not been identified yet. Therefore, we take a different approach here. Given a partial solution G'_F, we first construct a graph $G'^r_F = (A^r_p, E^r)$ for the unlinked activities except a_l, where $A^r_p = A_p \setminus (V(G'_F) \cup \{a_l\})$ and $E^r = \{(i,j) \in E | i, j \in A^r_p\}$. Then, G'_F is propagated on a sample x to obtain a partial schedule $S(G'_F, x)$. Next, we find the earliest precedence and resource feasible start time s_l for a_l based on $S(G'_F, x)$, and compute the duration $d_l(s_l, x)$ using Equation (6). Finally, we obtain a complete schedule by propagating G'^r_F on x based on $S(G'_F, x)$, $c_l = s_l + d_l(s_l, x)$, and a set of precedence relations $E^l = \{(i,j) \in E | i \notin A^r_p, j \in A^r_p\}$. Denote this schedule as $S(G'_F, a_l, x)$, then for a sample x, $MS(G'_F, a_l, x)$ is a lower bound for the makespan of any $\bar{G}'_F$ obtained by incorporating a_l to G'_F using a feasible link lk. The reason is that $\forall S(\bar{G}'_F, x)$, a_l cannot start earlier than s_l. According to Lemma 1, $MS(G'_F, a_l, x) \leq MS(\bar{G}'_F, x)$. By conducting the above process on each sample x, we have

$$LB(G'_F, a_l) = \frac{1}{M} \sum_{\xi=1}^{M} MS(G'_F, a_l, x^\xi). \tag{12}$$

6.4 Branching Heuristics

The general intuition in designing the branching heuristics is that we want to find high quality solutions as early as possible to prune more search space. Therefore, for the activity choosing step in Line 9 of Algorithm 1, we adopt two commonly used priority rules from the deterministic RCPSP for selecting activities, Maximum Total Successors (MTS) and Minimum Latest Finish Time (MLFT) [16]. These two rules often produce good solutions with heuristic schedule generation schemes [16] that are similar to our approach without backtracking. MTS prefers the activity with more immediate successors, while MLFT gives priority to the activity with smaller LFT value. Here the computation of MTS is the same as that of deterministic RCPSP since the original precedence relations are not affected by the workability uncertainty. On the other hand, we compute the LFT value for each activity by propagating the original project network G on a randomly generated sample x, which takes the workability uncertainty into consideration.

For the link choosing step in Line 5 of Algorithm 2, we design two heuristics, Minimum Average Earliest Start Time (MAEST) and Minimum Link Predecessors (MLP). The intuition of MAEST is based on Lemma 1. Specifically, for each link, MAEST computes the earliest start time $est(lk, x^\xi)$ of the chosen activity on each x^ξ, and prefers the one with smaller average value $1/M \cdot \sum_{\xi=1}^{M} est(lk, x^\xi)$. MLP is designed based on Observation 1, which prefers a link with fewer edges, i.e. a smaller value of $|V(lk)|$.

7. EMPIRICAL EVALUATION

In this section, we conduct a series of experiments on benchmark problem instances using real-world workability distribution data to evaluate different configurations of our approach, and to compare our approach with the best general-purpose POS generation approaches in the literature.

7.1 Experiment Setting

To model the workability uncertainty, here we use a distribution dataset collected from a real-world aero engine testing project. As shown in Figure 1, this dataset describes the Probability of Workability (POW) of four types of activities in each month of a year. To obtain a sample of workability from this distribution, we conduct a procedure that determines the workability of each time slot (day) in the horizon using a random sampling according to the corresponding POW value. We use this method to generate samples for both the SAA problem and the objective evaluation.

The RCPSP instances used in the experiments are generated using a benchmark problem generator *RanGen2* [30]. Besides the number of activities N and resources K, several parameters need to be specified, including order strength (OS), resource factor (RF) and resource-constrainedness (RC). OS specifies the project network G, and a lower OS indicates that more activities can be executed in parallel. RF and RC measure the resource utilization; a higher RF means that activities have more non-zero resource requirements b_{ik}, while a higher RC means that activities require more resources, i.e. b_{ik} is closer to c_k. More details of these parameters can be found in [30]. Two sets of instances (Set1 and Set2) are generated in our experiments. Below we describe Set1 which will be used in both Section 7.2 and 7.3, while details of Set2 will be discussed in Section 7.3. In Set1, the values of N and K are chosen from {10,20,30} and {1,2,3}, respectively, while the values of OS, RF and RC are chosen from {0.2,0.7} to represent the "low" and "high" level. For each combination of these five parameters, we generate a subset of 10 instances, therefore Set1 consists of 720 instances in total. For each activity in these instances, we randomly assign a type $z \in \{1, 2, 3, 4\}$. In addition, we increase the deterministic activity durations d^s_i 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.

In our experiments, all algorithms are implemented using JAVA 1.8, and runs on an Intel Xeon Workstation (3.5GHz, 16GB). We limit the CPU time of our branch-and-bound algorithm to 300 seconds, and use the best solution found. As mentioned in Section 5, it is intractable to compute the exact value of $g(G_F)$. Hence, we use Monte Carlo simulation to compute an estimated value $\hat{g}_{M_s}(G_F)$, which is to generate a set of M_s testing samples and compute the sample average function in Equation (5). This is a reliable way for estimating an expected value, when M_s is large [15]. Here we choose $M_s = 2000$ as in [15].

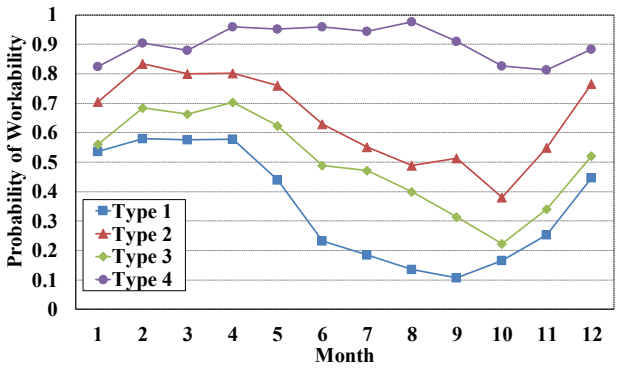

Figure 1: Monthly probability of workability

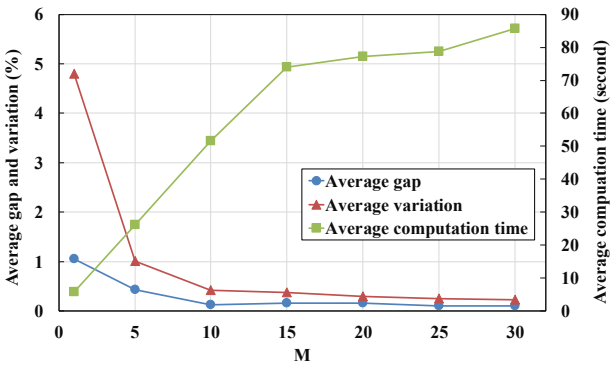

Figure 2: Impact of sample size

7.2 Performance of our Approach

This section summarizes the results for evaluating our approach. Instances in Set1 will be used in this section.

7.2.1 Impact of Sample Size

Sample size M is an important parameter that affects the performance of SAA. Intuitively, a larger M produces better solutions, but requires longer computation time. Here we evaluate the impact of M using the optimality gap estimator σ in [15] as the criterion for solution quality. Suppose M_r replications of SAA problems are solved independently, i.e. each problem η is solved on its own sample set $\boldsymbol{x}_\eta$ to obtain a solution G_F^η, with the objective $\hat{g}(G_F^\eta)$ on $\boldsymbol{x}_\eta$. Denote $G_F^{\eta*}$ as the solution with the lowest objective value, then $\sigma = 1/M_r \sum_{\eta=1}^{M_r} |\hat{g}(G_F^\eta) - \hat{g}_{M_s}(G_F^{\eta*})|$. In addition, the variance of σ is computed as $var_\sigma = S_{M_s}^2/M_s + S_{M_r}^2/M_r$, where $S_{M_s}^2$ and $S_{M_r}^2$ are the variance of the objective values in M_s times of simulations and M_r times of SAA replications, respectively. Here we choose $M_r = 20$. For clarity and brevity, we report the results on a representative instance subset (contains 10 instances), while a similar phenomenon is observed in other instances. For each instance in this subset, we normalize σ and var_σ using $\hat{g}_{M_s}(G_F^{\eta*})$, and plot their average percentage values in Figure 2, along with the average computation time. The curves in Figure 2 clearly show the trade-off between solution quality and computational effort. In general, both the average value and variance of σ become stable when $M \geq 20$. Therefore, we set $M = 20$ in the remaining experiments.

7.2.2 Comparison of Branching Heuristics

In Section 6.4, we propose several heuristics for choosing the next branching alternative in the activity and link levels. To evaluate their performance, here we run the branch-and-bound algorithm with all the four possible configurations, i.e. MLFT+MAEST, MLFT+MLP, MTS+MAEST, and MTS+MLP. Specifically, when one heuristic is used, the other one for the same level will be used for tie-breaking. We report the performance of these configurations on 240 instances from Set1 with $N = 20$ in Table 1, classified according to the other four parameters. We can conclude that in general, MLFT and MAEST give better performance than MTS and MLP, respectively. A possible explanation is that MLFT and MAEST are more "focused" than MTS and MLP in optimizing our objective function, since they are time-related criteria while the other two are based on the graph structure of POS. In the remaining experiments, we use MLFT+MAEST as the branching heuristics.

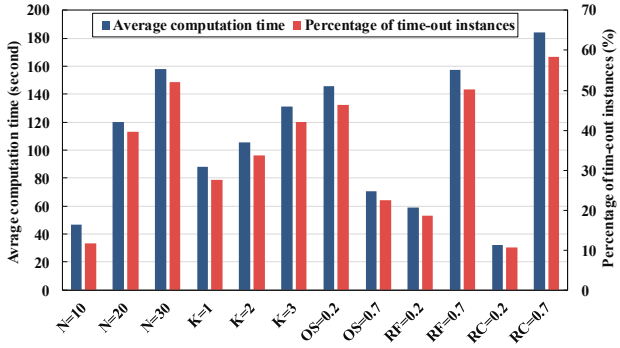

Figure 3: Impact of problem parameters

7.2.3 Impact of Problem Parameters

In the experiments, our branch-and-bound algorithm is able to find the optimal solutions for 65.6% of the instances in Set1 (472 out of 720), with an average computation time of 108.3 seconds. In this section, we examine the computational efficiency of our algorithm against the problem parameters, including N, K, OS, RF and RC. We classify the 720 instances according to the parameter values, and plot the average computation time and the percentage of time-out instances in Figure 3. As shown in this figure, the hardness for solving an instance increases with N, K, RF and RC, but decreases with OS. Below we give a brief analysis. For N and K, it is easy to see that the problem size grows with these two parameters. For RF and RC, larger values indicate that an activity tends to require more types of resources with higher demands, which leads to a larger search space since more branching alternatives exist in the link level of the algorithm. For OS, a higher value means that the project network G contains more precedence constraints, and in this case the search space is smaller since the number of branching alternatives in the activity level tends to decrease. Aside from the above observation, we can also see that our algorithm seems to be more sensitive to the resource related parameters RF and RC, which implies that the number of link level branching alternatives, i.e. $|LK|$, may give more impact on the computational efficiency.

7.3 Comparison with other Approaches

In this section, we compare the quality of solutions produced by our approach with the ones generated by the best general-purpose approaches (those do not exploit a prior stochastic knowledge). Specifically, we implement another

Table 1: Comparison of branching heuristics

Instance	MLFT+MAEST				MLFT+MLP				MTS+MAEST				MTS+MLP			
group	Best[1]	First[2]	Time[3]	PTO[4]	Best	First	Time	PTO	Best	First	Time	PTO	Best	First	Time	PTO
K=1	575.41	**586.9**	92.65	30.0	576.34	626.18	92.68	30.0	**571.96**	589.25	92.89	30.0	574.71	653.21	**89.14**	**28.75**
K=2	634.74	**654.1**	**120.4**	**40.0**	**632.5**	692.82	120.58	40.0	634.36	658.26	131.69	43.75	633.82	700.29	127.93	42.5
K=3	**672.6**	**688.0**	147.7	**48.75**	685.68	774.88	151.41	50.0	683.13	702.16	**146.98**	48.75	698.39	793.23	154.55	51.25
OS=0.2	**813.6**	**838.9**	**150.6**	**50.0**	821.33	937.24	153.26	50.83	820.65	853.42	165.66	55.0	832.07	972.79	165.65	55.0
OS=0.7	441.62	447.1	89.93	29.17	441.7	458.68	89.85	29.17	**438.98**	**446.37**	**82.04**	**26.67**	439.21	458.36	82.09	26.67
RF=0.2	469.3	**477.3**	57.74	**19.17**	**469.28**	491.21	57.75	19.17	470.22	481.66	**57.72**	19.17	470.25	495.94	57.73	19.17
RF=0.7	**785.9**	**808.7**	**182.8**	**60.0**	793.76	904.71	185.36	60.83	789.41	818.13	189.98	62.5	801.03	935.21	190.02	62.5
RC=0.2	**531.9**	**535.2**	**33.23**	**10.83**	537.46	640.07	35.82	11.67	536.39	541.14	42.82	14.17	545.21	666.38	42.83	14.17
RC=0.7	723.29	**750.7**	207.28	68.33	725.58	755.86	207.29	68.33	**723.24**	758.65	**204.88**	**67.5**	726.07	764.77	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.
[4] The percentage (%) of time-out instances.

Table 2: Quality of partial-order schedules

	Results on Set1								Results on Set2							
Instance	BnB	ESTA-Iter		EBA-Minflow		Artigues03			Instance	BnB	ESTA-Iter		EBA-Minflow		Artigues03	
group	AvgObj[1]	AvgObj	Diff(%)[2]	AvgObj	Diff(%)	AvgObj	Diff(%)		group	AvgObj	AvgObj	Diff(%)	AvgObj	Diff(%)	AvgObj	Diff(%)
N=10	**699.36**	722.31	3.18	744.01	6	743.07	5.88		N=10	**897.02**	996.22	9.96	1127.67	20.45	1028.67	12.8
N=20	**627.75**	654.82	4.13	722.88	13.16	678.14	7.43		N=20	**941.3**	1079.82	12.83	1333.94	29.43	1170.69	19.59
N=30	**639.25**	656.47	2.62	754.91	15.32	683.71	6.5		N=30	**992.41**	1117.48	11.19	1513.08	34.41	1185.34	16.28
K=1	**608.62**	615.6	1.13	673.94	9.69	644.27	5.53		K=1	**852.64**	957.02	10.91	1213.79	29.75	1021.23	16.51
K=2	**646.7**	667.45	3.11	733.08	11.78	694.71	6.91		K=2	**937.84**	1070.7	12.41	1340.17	30.02	1131.93	17.15
K=3	**711.04**	750.55	5.26	814.69	12.72	765.98	7.17		K=3	**1040.23**	1165.79	10.77	1420.72	26.78	1231.53	15.53
OS=0.2	**842.39**	887.06	5.04	1001.15	15.86	921.62	8.6		OS=0.2	**1119.1**	1369.51	18.28	1754.99	36.23	1408.78	20.56
OS=0.7	**468.51**	468.67	0.03	479.28	2.25	481.09	2.62		OS=0.5	**759.5**	768.05	1.11	894.8	14.17	847.68	9.39
RF=0.2	**512.16**	514.44	0.44	520.2	1.55	522.35	1.95		RF=0.7	**874.67**	968.65	9.7	1210.94	27.77	1031.08	15.17
RF=0.7	**798.75**	841.29	5.06	961.58	16.93	881.48	9.39		RF=0.9	**1012.47**	1160.36	12.75	1438.86	29.63	1225.38	17.37
RC=0.2	**548.6**	573.08	4.27	695.12	21.08	599.52	8.49		RC=0.2	**750.16**	826.36	9.22	1142.79	34.36	911.96	17.74
RC=0.7	**762.31**	782.65	2.6	786.17	3.04	804.09	5.2		RC=0.4	**1136.98**	1302.65	12.72	1507	24.55	1344.5	15.43

[1] The average of the objective values.
[2] The difference (%) of the average objective value from that of the branch-and-bound algorithm.

three algorithms, including EBA with the min-flow based resource conflict detection (EBA-Minflow) [21], ESTA^{C} with iterative chaining (ESTA-Iter) [26], and a schedule generation scheme based method (Artigues03) in [3]. On the 720 instances from Set1, our approach is able to give the best results on 99.3% (715 out of 720) instances. In the left part of Table 2, we list the results of the four approaches on the instances in Set1, grouped by the problem parameters. As shown in this table, our approach outperforms the other three on all groups, which clearly shows the advantage of exploiting the stochastic knowledge about the workability uncertainties. Another observation is that EBA-Minflow tends to give worse performance than ESTA-Iter and Artigues03. This is because EBA based approaches generate a feasible POS by iteratively identifying and removing potential resource conflicts, but gives little attention to the precedence constraints in the original project graph G. On the other hand, ESTA-Iter tries to minimize the "dependencies" between activities, which can reduce the number of edges in the generated POS. This observation verifies our intuition of designing the branching heuristic MLP.

Another observation from the results in the left part of Table 2 is that the improvement of our approach tends to be lower on instances with higher OS, lower RF and higher RC. Below we give an intuitive explanation. For a higher OS, the original project graph G is denser, thus a large portion of the edges in the POS is already determined by E. For a lower RF, the activities tend to require less resources, which also leads to a POS where a majority number of edges are from E. For a higher RC, a possible reason is that the search space is larger, hence our approach cannot find the optimal solutions within the time limit. To further examine the performance of our approach on lower OS, higher RF and lower RC, we generate another instance set (Set2) which is more focused on these three parameters. Set2 is generated using the same process described in Section 7.1, with $N \in \{10, 20, 30\}$, $K \in \{1, 2, 3\}$, OS$\in \{0.2, 0.5\}$, RF$\in \{0.7, 0.9\}$, and RC$\in \{0.2, 0.4\}$. Results on these 720 instances are summarized in the right part of Table 2, which shows a more prominent improvement compared to other approaches. To summarize, by exploiting the stochastic knowledge, our approach can generate POS with lower expected makespan than general-purpose approaches.

8. CONCLUSION AND FUTURE WORK

This paper studies the problem of proactive project scheduling under time-dependent workability uncertainty. We employ SAA to tackle the hard stochastic optimization problem, and propose a branch-and-bound algorithm to solve the resulting SAA problem to find the partial-order solution that minimizes the expected makespan. Experiment results show the effectiveness of our approach. The presented approach will be implemented as an intelligent scheduling agent in an agent-based simulation platform for complex business processes. For future work, an immediate direction is to improve the computational efficiency, by designing more effective lower bounds and introducing dominance rules that can prune a large portion of the search space. Another direction is to extend our approach to other objectives, such as solution stability and robustness, which are also considered as practical objectives as the expected makespan.

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.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice hall, 1993.

[2] O. Alagöz and M. Azizoğlu. Rescheduling of identical parallel machines under machine eligibility constraints. *European Journal of Operational Research*, 149(3):523–532, 2003.

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

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

[5] J. Bidot, T. Vidal, P. Laborie, and J. C. Beck. A general framework for scheduling in a stochastic environment. In *IJCAI*, pages 56–61, 2007.

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

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

[8] J. Cui, P. Yu, C. Fang, P. Haslum, and B. C. Williams. Optimising bounds in simple temporal networks with uncertainty under dynamic controllability constraints. In *ICAPS*, pages 52–60, 2015.

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

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

[11] H. El Sakkout and M. Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, 5(4):359–388, 2000.

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

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

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

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

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

[17] P. Laborie. Complete mcsv-based search: Application to resource constrained project scheduling. In *IJCAI*, pages 181–186, 2005.

[18] P. Lamas and E. Demeulemeester. A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations. *Journal of Scheduling*, pages 1–20, 2015.

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

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

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

[22] X. Mao, N. Roos, and A. Salden. Stable multi-project scheduling of airport ground handling services by heterogeneous agents. In *AAMAS*, pages 537–544, 2009.

[23] S. V. Mehta and R. M. Uzsoy. Predictable scheduling of a job shop subject to breakdowns. *IEEE Transactions on Robotics and Automation*, 14(3):365–378, 1998.

[24] P. Morris and N. Muscettola. Temporal dynamic controllability revisited. In *AAAI*, pages 1193–1198, 2005.

[25] P. Morris, N. Muscettola, and T. Vidal. Dynamic control of plans with temporal uncertainty. In *IJCAI*, pages 494–499, 2001.

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

[27] N. Policella, S. F. Smith, A. Cesta, and A. Oddi. Generating robust schedules through temporal flexibility. In *ICAPS*, volume 4, pages 209–218, 2004.

[28] W. Song, D. Kang, J. Zhang, and H. Xi. Decentralized multi-project scheduling via multi-unit combinatorial auction. In *AAMAS*, pages 836–844, 2016.

[29] P. Ströhle, E. H. Gerding, M. M. de Weerdt, S. Stein, and V. Robu. Online mechanism design for scheduling non-preemptive jobs under uncertain supply and demand. In *AAMAS*, pages 437–444, 2014.

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

[31] P. Varakantham, N. Fu, and H. C. Lau. A proactive sampling approach to project scheduling under uncertainty. In *AAAI*, 2016.