The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

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

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

In this paper, we consider the challenging problem of risk-aware proactive scheduling with the objective of minimizing robust makespan. State-of-the-art approaches based on probabilistic constrained optimization lead to Mixed Integer Linear Programs that must be heuristically approximated. We optimize the robust makespan via a coherent risk measure, Conditional Value-at-Risk (CVaR). Since traditional CVaR optimization approaches assuming linear spaces does not suit our problem, we propose a general branch-and-bound framework for combinatorial CVaR minimization. We then design an approximate complete algorithm, and employ resource reasoning to enable constraint propagation for multiple samples. Empirical results show that our algorithm outperforms state-of-the-art approaches with higher solution quality.

1 Introduction

Real-world scheduling applications often face dynamic situations due to the existence of considerable amounts of uncertainty, such as equipment breakdowns, delays of certain tasks, unforeseen weather conditions, etc. Therefore, practical scheduling approaches should take uncertainty into account. Specifically, the scheduling problem we consider is the challenging Resource-Constrained Project Scheduling Problem (RCPSP) with uncertain activity durations. In line with (Beck and Wilson 2007; Fu et al. 2012; Varakantham, Fu, and Lau 2016; Fu, Varakantham, and Lau 2016), we investigate proactive scheduling approaches for optimizing a risk-aware objective, i.e. minimization of the α-robust makespan, which focuses on controlling the probability that the actual makespan exceeds the α-robust makespan within a predefined risk parameter $\alpha \in (0, 1)$. Compared to the expected makespan as adopted in (Lombardi, Milano, and Benini 2013; Creemers 2015; Song et al. 2017), α-robust makespan is more practical since the actual makespan may be much worse than the expected value with a high chance.

The incorporation of uncertainty brings additional challenges to the already intractable RCPSP, since it is hard to even evaluate a solution (Beck and Wilson 2007). Existing approaches often resort to sampling-based techniques to mitigate the complexity. For example, Beck and Wilson (2007) and Fu et al. (2012) use sampling and simulation to evaluate solutions. State-of-the-art approaches (Varakantham, Fu, and Lau 2016; Fu, Varakantham, and Lau 2016) consider α-robust makespan minimization as a probabilistic constrained problem, which can be approximated by Sample Average Approximation (SAA) with the guarantee of converging to the optimal solution (Luedtke and Ahmed 2008). However, these approaches result in Mixed Integer Linear Programs (MILP) which are computationally prohibitive even with sophisticated solvers. To scale up the MILPs, the summarization heuristic is applied to aggregate the samples into a representative one. However, this heuristic compromises the convergence guarantee, and decreases solution quality.

In this paper, we propose to optimizing the α-robust makespan via Conditional Value-at-Risk (CVaR), a popular measure in risk-sensitive applications (Rockafellar and Uryasev 2002). Our approach scales up to hundreds of samples without the need of sample summarization, hence can provide better robust makespan and better control of the risk parameter α. Based on the expectation form of CVaR minimization, we approximate the proactive problem using SAA. However, we show that the resulting problem is NP-hard due to the combinatorial nature of RCPSP. This also excludes the traditional CVaR minimization approaches that assume the decision space is linear (Hong, Hu, and Liu 2014). Thus, we design a general branch-and-bound framework for CVaR minimization in combinatorial space. We then instantiate this framework to develop a branch-and-bound algorithm based on constraint propagation and the network flow theory, by extending the related components in (Leus and Herroelen 2004) designed for single resource problems. Our algorithm shares similarities with the Precedence Constraint Posting (PCP) approaches in (Laborie 2005; Lombardi, Milano, and Benini 2013). However, existing PCP approaches are designed for deterministic RCPSP where temporal reasoning can be applied for branching and constraint propagation. In contrast, our problem is built on multiple duration samples, which makes temporal reasoning very difficult. Thereby, our algorithm purely reasons with resource constraints, except the lower bound computation.

Finally, we conduct extensive experiments on benchmark instances and commonly used uncertainty models. The results show that our approach scales well to a large number of samples, and can produce solutions with lower α-robust

makespan than state-of-the-art approaches, especially when the uncertainty level is high.

2 Preliminaries

This section introduces the basic concepts and notations.

2.1 Minimization of VaR and CVaR

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

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

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

$$F_\beta(x, \omega) = \omega + \frac{1}{1-\beta}\mathbb{E}\{[g(x, \boldsymbol{y}) - \omega]^+\}, \quad (1)$$

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

$$(x^*, \omega^*) = \underset{(x,\omega)\in X\times\mathbb{R}}{\operatorname{argmin}} F_\beta(x, \omega), \quad (2)$$

where x^* minimizes CVaR. Since F_β has an expectation form, Sample Average Approximation (SAA) (Kleywegt, Shapiro, and Homem-de Mello 2002) is immediately applicable to approximate Problem (2), by optimizing $\hat{F}_\beta = \hat{F}_\beta(x, \omega)$ defined below in the joint space $X \times \mathbb{R}$:

$$\hat{F}_\beta(x, \omega) = \omega + \frac{1}{(1-\beta)Q}\sum_{q=1}^{Q}[g(x, y^q) - \omega]^+, \quad (3)$$

where $(y^1, ..., y^Q)$ is Q samples independently drawn from $\boldsymbol{y}$. Guaranteed by the property of SAA, the optimal solution $(\hat{x}^*, \hat{\omega}^*)$ is proven to converge to (x^*, ω^*) in Equation (2) exponentially fast as the increase of sample size Q.

2.2 RCPSP with Uncertain Durations

Deterministic RCPSP involves N non-preemptive activities $A = \{a_1, ..., a_N\}$ and K renewable resources $R = \{r_1, ..., r_K\}$. Throughout its fixed duration d_i^0, each $a_i \in A$ requires $b_{ik} \in \mathbb{N}$ units of $r_k \in R$ with limited capacity $C_k \in \mathbb{N}$. Two dummy activities a_0 and a_{N+1} with zero durations are often added to represent project start and finish. Precedence constraints (i, j) could be imposed on any two activities $a_i, a_j \in A_P = A \cup \{a_0, a_{N+1}\}$, indicating a_j must start after the completion of a_i, i.e. $s_j \geq s_i + d_i^0$, where s_i is the start time of a_i. Let E_P be the set of all precedence constraints, then the temporal relations of the project can be represented using the Activity-On-Node (AON) network $G_P = (A_P, E_P)$, which is a Directed Acyclic Graph (DAG) with vertex set A_P and edge set E_P. Below we will use $V(G)$, $E(G)$, and $Tr(G)$ to represent the vertex set, edge set, and transitive closure of a graph G.

A (start-time) schedule is a vector $S = (s_0, ..., s_{N+1})$. Without loss of generality, we assume a_0 always starts at 0, then the makespan is $MS(S) = s_{N+1}$. A schedule is feasible if no precedence or resource constraint is violated. A feasible schedule is optimal if it has the minimal makespan. Under duration uncertainty, each a_i has a random duration $\boldsymbol{d}_i$ instead of a fixed one. Let $\boldsymbol{d} = (\boldsymbol{d}_1, ..., \boldsymbol{d}_N)$ be the vector of random durations, and $d = (d_1, ..., d_N)$ be a realization of $\boldsymbol{d}$. Since d_i may not equal to d_i^0, a feasible start-time schedule may become infeasible in execution. Alternatively, a project can also be executed following a partial order schedule (POS), a flexible policy that determines activity start times during execution.

A POS $G_H = (A_P, E_P \cup E_H)$ is an augmented DAG of G_P with additional precedence constraints in E_H, such that any temporal feasible schedule is resource feasible (Policella et al. 2004). In other words, all possible resource conflicts are removed by the edges in E_H. Hence the start time of each a_i can be determined at the execution time very easily:

$$s_i = \max\{s_j + d_j | (j, i) \in E(G_H)\}, \quad (4)$$

i.e. a_i starts right after the completion of all predecessors specified by G_H. Along with execution, a feasible schedule with respect to the actual durations d is obtained. Let $S(G_H, d)$ and $MS(G_H, d)$ be this schedule and its makespan. Since $\boldsymbol{d}$ is random, the makespan of G_H is also a random variable $MS(G_H, \boldsymbol{d})$. Let $\boldsymbol{G}_H$ be the set of POSs.

Another concept used here is the AON-flow Network (Artigues, Michelon, and Reusser 2003), which is also an augmented DAG $G_F = (A_P, E_P \cup E_F)$ of G_P, but $E_P \cap E_F$ is not necessarily $\emptyset$. For each $(i, j) \in E_F$, a flow vector $f_{ij} = (f_{ij1}, ..., f_{ijK})$ is associated to denote the amount of r_k transfered from a_i to a_j. Let the requirement for each r_k of the two dummy activities be $b_{0k} = b_{N+1,k} = C_k$. Then G_F should satisfy the below conditions: 1) Positive flow: $\sum_{r_k \in R} f_{ijk} > 0, \forall (i, j) \in E_F$; 2) Inflow balance: $\sum_{(j,i)\in E_F} f_{jik} = b_{ik}, \forall r_k \in R, a_i \in A_p \setminus \{a_0\}$; 3) Outflow balance: $\sum_{(i,j)\in E_F} f_{ijk} = b_{ik}, \forall r_k \in R, a_i \in A_p \setminus \{a_{N+1}\}$. Let $\boldsymbol{G}_F$ be the set of AON-flow Networks. Apparently, for any $G_F \in \boldsymbol{G}_F$, $G_H = (A_P, E(G_F))$ is a POS since the resource conflicts are resolved by the flows. In Section 5.1 we will show that we can determine if a DAG G_V is a POS, by checking if it can accommodate a feasible flow.

3 CVaR based Proactive Scheduling

As mentioned, we aim to optimize the α-robust makespan. Here we give its definition following the one in (Beck and Wilson 2007) for job-shop problems. Given a risk parameter $\alpha \in (0,1)$, a real value D is α-achievable for POS G_H if $Pr(MS(G_H, \boldsymbol{d}) \leq D) \geq 1-\alpha$, i.e. the probability that the random makespan exceeds D is bounded by α. Then the α-robust makespan of G_H is the minimum of all the α-achievable D, i.e. $D_\alpha(G_H) = \min\{D | Pr(MS(G_H, \boldsymbol{d}) \leq D) \geq 1-\alpha\}$. The proactive problem can be formulated as:

$$G_H^* = \operatorname*{argmin}_{G_H \in \boldsymbol{G}_H} D_\alpha(G_H). \tag{5}$$

It is easy to verify that $D_\alpha(G_H)$ is the β-VaR of G_H on $\boldsymbol{d}$, with $\beta = 1-\alpha$ and the loss being $MS(G_H, \boldsymbol{d})$. Hence Problem (5) is equivalent to minimizing the β-VaR.

The typical way for VaR minimization is to transform it into a chance constrained optimization problem (Hong, Hu, and Liu 2014). However, for Problem (5), this results in MILPs that are computationally prohibitive, as presented in (Varakantham, Fu, and Lau 2016; Fu, Varakantham, and Lau 2016). Therefore, we take a different approach here which optimizes CVaR instead of VaR. To be more specific, we optimize the approximate function $\hat{F}_\beta$ defined in Equation (3) on independent samples $\{d^1, ..., d^Q\}$ drawn from $\boldsymbol{d}$:

$$(\hat{G}_H^*, \hat{\omega}^*) = \operatorname*{argmin}_{(G_H, \omega) \in \boldsymbol{G}_H \times \mathbb{R}} \hat{F}_\beta(G_H, \omega). \tag{6}$$

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

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

Proposition 1. *The optimization problem (6) is NP-hard.*

Proof. Our proof follows the one for RCPSP in (Blazewicz, Lenstra, and Kan 1983), which reduces the NP-complete problem Partition Into Triangles (PIT) to RCPSP. The PIT problem is: given graph $G = (V, E)$ with $|V| = 3t$, can we partition G into t disjoint subsets, each containing three pairwise adjacent vertices? For any PIT instance, we construct an instance of Problem (6) as follows. First, we create a RCPSP as in (Blazewicz, Lenstra, and Kan 1983): for each $i \in V$, create an activity a_i; for each $(i,j) \notin E$, create a resource r_{ij} with $C_{ij} = 1$, which is consumed by a_i and a_j only, with a requirement of 1. Next, we add one sample d with $d_i = 1$ for all a_i, and obtain an instance of problem (6) with the objective $\hat{F}_\beta(G_H, \omega) = \omega + 1/(1-\beta)[MS(G_H, d) - \omega]^+$.

We claim that this problem has a solution (G_H, ω) with $\hat{F}_\beta(G_H, \omega) \leq t$ if and only if the corresponding PIT instance has a solution. Firstly, if the PIT instance has a solution, then a feasible schedule S can be obtained immediately with $MS(S) \leq t$. Then on each resource unit, we sort the consuming activities in ascending order based on their start times in S, and create a POS G_H by adding precedence constraints for each activity and its immediate successor on each resource it consumed. Clearly $MS(G_H, d) \leq t$, hence we have a solution (G_H, ω) with $\omega = MS(G_H, d)$ and $\hat{F}_\beta(G_H, \omega) = MS(G_H, d) \leq t$. Secondly, if problem (6) has a solution (G_H, ω) with $\hat{F}_\beta(G_H, \omega) \leq t$, then we must have $MS(G_H, d) \leq t$. This is because if $MS(G_H, d) = t' > t$, then function $\hat{F}_\beta(\omega) = \omega + 1/(1-\beta)[t' - \omega]^+$ has an infimum t', indicating $\hat{F}_\beta(G_H, \omega) > t$ for any $\omega \in \mathbb{R}$. Hence schedule $S = S(G_H, d)$ has a makespan $MS(S) \leq t$, indicating the PIT instance has a solution. □

Therefore, linear approaches cannot be applied to Problem (6). In the next section, we design a general branch-and-bound framework for combinatorial CVaR minimization.

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

In general, a branch-and-bound algorithm iteratively partitions the solution space into smaller pieces, and uses a bounding function to fathom searching in certain solution pieces. Here we aim at designing a branch-and-bound framework for minimizing $\hat{F}_\beta$ defined in Equation (3) in the solution space $X \times \mathbb{R}$ where X is combinatorial. Our framework only partitions X, since ω is an unbounded real variable which is relatively easy to optimize.

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

$$L_\beta(X', \omega) = \omega + \frac{1}{(1-\beta)Q} \sum_{q=1}^{Q} [g_{LB}(X', y^q) - \omega]^+. \tag{7}$$

Then we can have the following conclusion:

Proposition 2. *Define $LB(X')$ as follows:*

$$LB(X') = \min_{\omega \in \mathbb{R}} L_\beta(X', \omega), \tag{8}$$

then $LB(X')$ is a lower bound for X'.

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

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

According to Proposition 2, the lower bound value can be computed in two steps: 1) compute the lower bound on each sample; 2) solve the optimization problem (8). It is easy to verify that L_β is convex (though not differentiable) with respect to ω, therefore (8) is a univariate convex optimization problem, which can be solved by standard techniques such as the subgradient method. Nevertheless, we can show that Problem (8) can be solved more efficiently by simply ranking the sample lower bounds.

Proposition 3. *If Q sample lower bound values are ranked ascendingly as $g^1_{LB} \leq ... \leq g^Q_{LB}$, then $\omega^* = g^{k^*}_{LB}$ with $k^* = \lceil \beta Q \rceil$ solves Problem (8) optimally.*

Proof. The ranked sample lower bounds split $\mathbb{R}$ into a set of intervals $(-\infty, g^1_{LB}]$,...,$[g^k_{LB}, g^{k+1}_{LB}]$,...,$[g^Q_{LB}, \infty)$. It is easy to verify that L_β is linear in each interval, and is continuous in $\mathbb{R}$. We can then write the derivative of L_β with respect to ω: when $\omega \leq g^1_{LB}$, $L'_\beta = -\beta/(1-\beta) < 0$; for any integer $k \in [1, Q-1]$, when $\omega \in [g^k_{LB}, g^{k+1}_{LB}]$, $L'_\beta = (k-\beta Q)/(Q-\beta Q)$; when $\omega \geq g^Q_{LB}$, $L'_\beta = 1 > 0$. Hence, along with the increase of ω in $\mathbb{R}$, L'_β increases from negative to positive. The smallest k that makes $L'_\beta \geq 0$ is $k^* = \lceil \beta Q \rceil$, meaning that L_β stops decreasing in $[g^{k^*}_{LB}, g^{k^*+1}_{LB}]$ and $\omega^* = g^{k^*}_{LB}$ is an optimal solution to Problem (8). □

Therefore, $LB(X')$ can be obtained very easily after computing the sample lower bounds. With proper branching functions to partition the solution space X, the branch-and-bound process can be executed correctly to find the optimal solution $(\hat{x}^*, \hat{\omega}^*)$. Note that when a feasible decision $x' \in X$ is reached, a candidate (x', ω') for the optimal solution can be obtained by fixing the loss $g(x', y^q)$ in Equation (3) for each sample, and minimizing $\hat{F}_\beta$ with respect to ω following a similar ranking procedure as minimizing L_β. The sample losses can also be used to retrieve the (approximate) β-VaR of x' on the samples $(y^1, ..., y^Q)$, i.e. $\hat{\phi}_\beta(x') = \max\{g(x', y^q) | g(x', y^q) \leq \omega'\}$ (Rockafellar and Uryasev 2002). This framework is general and applicable for any combinatorial CVaR minimization problem, as long as the sample bounding function g_{LB} is available.

5 The Proactive Scheduling Algorithm

In this section, we instantiate our CVaR minimization framework to solve Problem (6). We first describe how to check if a given augmented DAG is a partial order solution (POS).

5.1 POS Checking

When there is only one resource r with capacity C, it has been shown in (Leus and Herroelen 2004) that for a given augmented DAG $G_V = (A_P, E_P \cup E_V)$, the existence of a feasible flow can be checked by computing a maximum flow in a transformed network G'_V constructed as follows: 1) create two vertices i^s and i^t for each $a_i \in A$, and one vertex for a_0 and a_{N+1} named as 0^s and $(N+1)^t$, respectively; 2) create two vertices, s and t with an edge (t, s) as the virtual source and sink, and add edges (s, i^s), (i^t, t) for all $a_i \in A_P$; 3) for each $(i, j) \in E(G_V)$, add an edge (i^s, j^t).

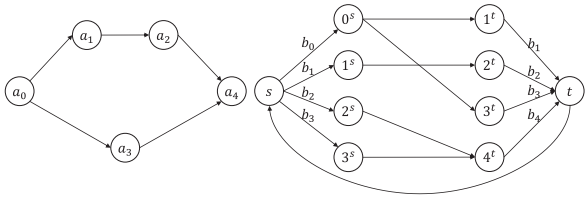

Figure 1: An example of network transformation (left: original DAG G_V; right: transformed network G'_V, where integers beside edges represent capacities)

Each (s, i^s) and (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 1. Let $f(G'_V)$ be the maximum (s, t) flow value in G'_V, then there exists an AON-flow Network G_F with $E(G_F) \subseteq E(G_V)$ if and only if $f(G'_V) = f_{max}$, where $f_{max} = C + \sum_{a_i \in A} b_i$. Moreover, a feasible flow in G_V can be obtained by setting f_{ij} to the flow value on the edge (i^s, j^t) in G'_V. Due to the integrality property of the maximum flow problem, all f_{ij} should be integers. The above procedure can be done efficiently using maximum flow algorithms (e.g. Edmonds-Karp algorithm).

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

Proposition 4. *For any POS $G_H \in \boldsymbol{G}_H$, there must be an AON-flow Network $G_F \in \boldsymbol{G}_F$ such that $E(G_F) \subseteq E(G_H)$.*

Proof. If no such AON-flow Network exists, then there must be a resource $r_k \in R$ with $f(G'^k_H) < f^k_{max}$. This means there must be some activity a_i which cannot secure enough amount of r_k by the edges in $E(G_H)$, since the flow in G'^k_H 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_H, which implies that potential precedence constraints are needed to resolve resource conflicts. □

5.2 Branching Scheme

Starting from G_P, our algorithm employs a depth-first branch-and-bound process to add edges to G_P until a POS is obtained[1]. Since a POS must be acyclic, the set of feasible edges that can be added is $FS = \{(i, j) \notin E(G_P) | (j, i) \notin Tr(G_P)\}$. For each $(i, j) \in FS$, we maintain lower bound f^L_{ijk} and upper bound f^U_{ijk} of the (integer) flow f_{ijk} that

[1]Note that when a POS G_H is reached, the algorithm can backtrack safely. Because for any G'_H with $E(G_H) \subseteq E(G'_H)$, $MS(G_H, d) \leq MS(G'_H, d)$ holds for any sample d, therefore $\min_{\omega \in \mathbb{R}} \hat{F}_\beta(G_H, \omega) \leq \hat{F}_\beta(G'_H, \omega)$ holds for any ω.

can be imposed on it for r_k, with $0 \leq f^L_{ijk} \leq f^U_{ijk}$. Initially, $f^L_{ijk} = 0$ and $f^U_{ijk} = \min\{b_{ik}, b_{jk}\}$. During searching, these bounds are tightened by constraint propagation which will be detailed in Section 5.3. The flow bounds are also imposed to the transformed networks: for G'^k_V transformed from G_V, for any $(i,j) \in E(G_V)$, the flow of r_k carried by edge (i^s, j^t) should be within $[f^L_{ijk}, f^U_{ijk}]$. 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 (i,j) while $sum^U_{ij} = 0$ indicates (i,j) cannot carry flow for any r_k. Based on the bound values and branching decisions, an edge $(i,j) \in ES$ 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 $(j,i) \in Tr(G_V)$ where G_V is the current partial solution.

The skeleton of our algorithm is described by the pseudo code in Algorithm 1. The first step is to check if the current partial solution G_V is a POS, using the procedure in Section 5.1. If true, meaning a feasible solution is reached, then we compute an optimal candidate (G_V, ω) using the ranking procedure in Section 4, and compare it with the incumbent (G^*_H, ω^*) to check if a better solution is found. Otherwise, G_V is not a POS and more edges need to be added, which leads the algorithm to a two-level branching scheme.

In the first level, an eligible edge $(i,j) \in FS$ is chosen for branching (Line 6), if it is undecided and does not conflict with G_V. The heuristic for choosing edge will be discussed in Section 5.4. Then the lower bound of adding (i,j) to G_V is computed to determine if the search path should be pruned or not (Line 7). If not, Algorithm 1 enters the second level where (i,j) is first included in G_V by imposing $f^L_{ijk} = 1$ for a chosen resource r_k, until all $r_k \in R$ have been tried, i.e. chooseResource returns $null$ (Lines 9-13). Then, (i,j) is banned by removing it from G_V and imposing $f^U_{ijk} = 0$ (which automatically imposes $f^L_{ijk} = 0$) for all $r_k \in R$ (Lines 14-16). When a lower (upper) bound needs to be tightened, a function propagateLB (propagateUB) is called in Line 11 (15) to maintain the consistency of the flow bounds. The search path is expanded by calling BnB if the propagation successes, otherwise it is pruned. The searching process starts by calling BnB$(G_P, null, null, +\infty)$. Upon termination, the β-VaR of the best solution is also returned as the α-robust makespan.

Lower Bound. In function computeLB (Line 7), we only need to compute the lower bounds of adding (i,j) to G_V on each sample d^q. To this end, we relax all resource constraints and compute $MS(\bar{G}_V, d^q)$ for each d^q as the sample lower bounds, where $\bar{G}_V = (A_P, E(G_V) \cup \{i,j\})$.

Algorithm 1: BnB$(G_V, G^*_H, \omega^*, \hat{F}^*_\beta)$

Input: G_V: current partial solution; G^*_H: current best POS; ω^*: the best ω with G^*_H; $\hat{F}^*_\beta$: current best objective

```
1  if G_V is a POS then
2      (G_V, ω) ←minimizeF(G_V);
3      if F̂_β(G_V, ω) < F̂*_β then
4          Update G*_H, ω*, and F̂*_β;
5  else
6      (i, j) ←chooseEdge(G_V);
7      if computeLB(G_V, i, j) < F̂*_β then
8          G_V ← (A_P, E(G_V) ∪ {(i, j)});
9          k ←chooseResource(G_V);
10         while k ≠ null do
11             if propagateLB(i, j, k)=true then
12                 BnB(G_V, G*_H, ω*, F̂*_β); restore();
13             k ←chooseResource(G_V);
14         G_V ← (A_P, E(G_V) \ {(i, j)});
15         if propagateUB(i, j)=true then
16             BnB(G_V, G*_H, ω*, F̂*_β); restore();
17 return;
```

5.3 Constraint Propagation

For single resource problems, Leus and Herroelen (2004) propose to maintain the flow bound consistency by conducting constraint propagation on the *remainder network* $G_R = (A_P, E_P \cup E_R)$, where $E_R = \{(i,j) \in ES | f^U_{ij} > 0\}$ is the set of edges not banned by the current branching decisions. For $(i,j) \in E(G_R)$, let $O_{ij} = \{(i,l) \in E(G_R) | l \neq j\}$ and $I_{ij} = \{(l,j) \in E(G_R) | l \neq i\}$ be the set of other edges in $E(G_R)$ 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 as:

$$f^L_{ij} = \max\left\{ f^L_{ij}, b_i - \sum_{(i,l)\in O_{ij}} f^U_{il}, b_j - \sum_{(l,j)\in I_{ij}} f^U_{jl} \right\} \quad (9)$$

$$f^U_{ij} = \min\left\{ f^U_{ij}, b_i - \sum_{(i,l)\in O_{ij}} f^L_{il}, b_j - \sum_{(l,j)\in I_{ij}} f^L_{jl} \right\} \quad (10)$$

Consistency can be achieved by updating bounds for all edges in $E(G_R)$ till no bound changes. The network G'_R transformed from G_R using the procedure in Section 5.1 is also used for detecting infeasibility in (Leus and Herroelen 2004). If $f(G'_R) < f_{max}$, then clearly the current branching decisions cannot lead to any AON-flow Network.

For our problem with multiple resources, we maintain the flow bounds independently for each r_k based on Equations (9) and (10). The branching decisions in the second level of Algorithm 1 enable the independent bound updates: when an edge (i,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 (i,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. Note that constraint propagation may *imply* that certain edges $(i,j) \notin E(G_P)$ should be included in G_V (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 transformed network G'^k_R for G_R, and try to maximize flows in G'^k_R and G'^k_V for r_k affected by constraint propagation. If $f(G'^k_R) < f^k_{max}$ or G'^k_V is an infeasible network, then according to Proposition 4, the current branching decisions cannot lead to any POS and a false value is returned.

5.4 Heuristics

Essentially, by adding edges to a partial solution G_V, we wish to increase the maximum flow in each G'^k_V to f^k_{max} so that a POS is obtained. Hence, we prefer the edge that can bring the largest increment for each $f(G'^k_V)$ so that a solution is reached as early as possible. Here we design a heuristic *Resource Score* to estimate the contribution that an eligible edge (i, j) could have for the flow increment as follows:

$$RS(i,j) = \sum_{r_k \in R} \left\{ RS_k(i,j) = \frac{f^R_{ijk}}{f^k_{max} - f(G'^k_V)} \right\}, \quad (11)$$

where RS_k is a normalized estimate for the contribution of (i, j) to r_k, with the nominator f^R_{ijk} being the flow for r_k on edge (i, j) in the remainder network G_R and the denominator being the current flow gap for G'^k_V to reach f^k_{max}. Function chooseEdge in Line 6 of Algorithm 1 returns the edge (i, j) with the highest RS value as the next branching choice, and chooseResource in Line 9 returns the unexplored r_k with the highest RS_k value for (i, j).

6 Experimental Results

In this section, we empirically evaluate our approach on benchmark instances, and compare with two state-of-the-art approaches SORU-H (Varakantham, Fu, and Lau 2016) and BACCHUS (Fu, Varakantham, and Lau 2016). SORU-H computes start-time schedule, while BACCHUS generates POS as our approach does. Our algorithm is implemented in JAVA 1.8, while SORU-H and BACCHUS are coded using Java API for CPLEX 12.7.1. All algorithms run on an Intel Xeon E5 Workstation (3.5GHz, 16GB).

We generate RCPSP instances using RanGen2 (Vanhoucke et al. 2008), which requires five parameters: numbers of activities N and resources K, order strength OS, resource factor RF, and resource-constrainedness RC.[2] OS specifies the density of G_P, RF describes the average number of resources required per activity, and RC indicates the average fraction of resource capacities consumed per activity. N and K are chosen from {10, 20, 30} and {1, 2, 3} respectively, while RF, OS, and RF are chosen from {0.2, 0.7} which denote the "low" and "high" levels. For each parameter configuration, we generate 10 instances as a subset, hence totally 720 instances are generated. Two distributions are used to model the uncertainty: 1) a normal distribution $\mathcal{N}(d^0_i, \sigma^2)$ with d^0_i being the deterministic duration of a_i (an integer in [1, 10] in the generated instances) and $\sigma = 0.5$, which is used in (Fu et al. 2012; Varakantham, Fu, and Lau 2016); 2) an exponential distribution $Exp(1/d^0_i)$ used in (Creemers 2015; Leus and Herroelen 2004). The uncertainty level of Exp is higher than $\mathcal{N}$, since its squared coefficient of variance (SCV) is much higher (Creemers 2015)[3].

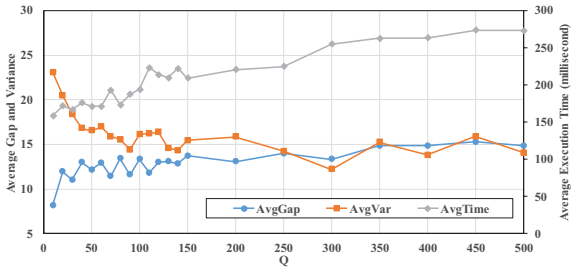

Figure 2: Results for sample size test

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

6.1 Analysis of Our Algorithm

We first examine our algorithm against different Q and α.

Impact of sample size. Since we (approximately) optimize F_β, we evaluate the impact of Q on F_β based on the gap estimator ρ of SAA (Kleywegt, Shapiro, and Homem-de Mello 2002). Specifically, we solve problem (6) $Q_r = 20$ times independently, and let (G^r_H, ω^r) be the solution with the lowest $\hat{F}_\beta$. Then ρ is estimated as $\rho = |\hat{F}'_\beta - \hat{F}^r_\beta(G^r_H, \omega^r)|$, where $\hat{F}'_\beta$ is the average objective of the Q_r replications, and $\hat{F}^r_\beta(G^r_H, \omega^r)$ is computed on Q_t testing samples. The variance of ρ is estimated as $var_\rho = S^2_r/Q_r + S^2_t/Q_t$, where S^2_r and S^2_t are variances of the Q_r objectives and the Q_t values of function $\omega^r + 1/(1-\beta)[g - \omega^r]^+$, respectively.

We plot the average gap, variance and execution time of our algorithm on a representative subset (10 instances) with Exp and $\alpha = 0.2$ in Figure 2, which clearly shows the trade-off between solution quality and computational effort. This is an expected observation, which is theoretically guaranteed by the properties of SAA. As shown, while the gap is relatively stable, its variance drops with the increase of Q, indicating the solution becomes more stable. The increase of execution time is not vary fast, which shows good scalability of our approach on large Q. Intuitively, this is because the samples are only used for lower bound computation which is very efficient in our CVaR minimization framework. We also

[2]Here we use this test set instead of public benchmarks such as PSPLIB, because we intend to evaluate the performance of our algorithm against different problem parameters, especially the resource-related ones (K, RF and RC) since our algorithm mainly reasons on the resource constraints.

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

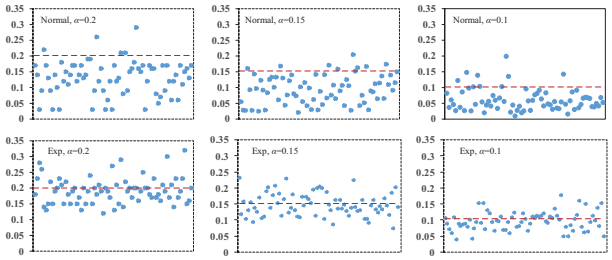

Figure 3: PoF distributions for different risk levels

have similar observations in other instance subsets. Here we use this instance set for illustration because all instances in this set are solved optimally in all replications for all Q values, which is more ideal for computing the gap estimator. In the remaining experiments, we set $Q = 100$.

Table 1: Results for different risk levels

	$\mathcal{N}$				Exp			
α	α-RM	PoF	#Vio [1]	#Vio-ϵ [2]	α-RM	PoF	#Vio	#Vio-ϵ
0.2	65.74	0.13	5	0	88.36	0.19	21	4
0.15	65.86	0.09	8	0	92.76	0.14	26	2
0.1	66.72	0.06	6	0	100.79	0.09	24	1

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

Impact of risk parameter. To examine the impact of α, we select 72 instances by randomly picking one in each subset. In Table 1, we present the average α-RM and PoF for the two distributions, with different risk levels $\alpha \in \{0.2, 0.15, 0.1\}$. We can observe that with stricter risk requirement (smaller α), α-RM increases since it needs to tolerate more execution scenarios. The increase of α-RM is higher for Exp than $\mathcal{N}$, since the uncertainty level of the former is higher. On average, PoF is close to α which shows a precise risk control. We plot the PoFs of the 72 instances in Figure 3, which shows most of PoFs are below the required level α. But still, some instances have higher PoF than α, as shown in the columns "#Vio" of Table 1, and Exp has more violations than $\mathcal{N}$. This is because Problem (6) is built on limited samples which cannot cover all situations. This is also observed in (Luedtke and Ahmed 2008; Varakantham, Fu, and Lau 2016), and they propose to solve the SAA problems with stricter risk level α' than required, i.e. $\alpha' = \alpha - \epsilon$. Following this idea, we set $\epsilon = 0.05$ for our algorithm, as can be observed in the columns "#Vio-ϵ" in Table 1 that this value can effectively reduce PoF violations.

6.2 Comparison with other Approaches

We first tune the parameter ϵ for SORU-H and BACCHUS. α is set to 0.2 in this section. We conduct experiments on the 72 instances used in Section 6.1, with $\epsilon \in \{0, 0.05, 0.1\}$. We report the number of violations in Table 2. As shown, $\epsilon = 0.05$ is reasonable for BACCHUS with a violation ratio of at most $12/72 = 16.7\%$ for both $\mathcal{N}$ and Exp. For SORU-H, $\epsilon = 0.1$ leads to satisfying results for $\mathcal{N}$, which is also the recommended value in (Varakantham, Fu, and Lau 2016).

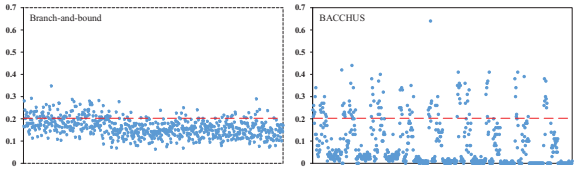

Figure 4: PoF distributions for Exp

But for Exp which has a higher uncertainty level, all test instances are violated with $\epsilon = 0.1$. This is because SORU-H generates start-time schedule as proactive solution, which is too rigid and has a high chance to violate when the duration uncertainty level is high. In contrast, BACCHUS and our approach generate flexible solution POS, which provides better robustness (Bidot et al. 2009). In the remaining experiments, we only report the results of SORU-H on $\mathcal{N}$.

Table 2: Number of violations for different ϵ values

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

Table 3: Summary of results

	$\mathcal{N}$			Exp	
	Ours	BACCHUS	SORU-H	Ours	BACCHUS
PoF$\leq \alpha$ (%)	98.06	98.19	86.94	82.08	84.44
α-RM [1]	64.66	65.85	68.4	102.97	133.12
LowestRM (%) [2]	77.81	26.99	7.78	89.71	14.12

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

We then execute the three algorithms on all the 720 instances, and summarize the results in Table 3. We say an instance test is successful, if its PoF$\leq \alpha$. For distribution $\mathcal{N}$, BACCHUS and our algorithm succeed on over 98% of the instances, which are more than SORU-H. On the 604 instances that are successful for all three algorithms, the average α-RM values are comparable. However, our algorithm achieves the lowest α-RM in over 77% of the successful instances, which is significantly higher than the other two. For distribution Exp, BACCHUS and our algorithm are able to succeed on over 80% of the instances. However, on the 517 instances successfully solved by both algorithms, the average α-RM produced by our algorithm is significantly lower than that of BACCHUS, with a 25% improvement. In fact, our algorithm achieves lower α-RM on nearly 90% of the successful instances. We believe this performance gap is caused by the summarization heuristic used in BACCHUS. To verify our intuition, we plot the 720 PoF values produced by BACCHUS and our algorithm with Exp in Figure 4. As shown, PoF values of our algorithm distribute densely around the required level $\alpha = 0.2$, with over 93% PoFs within [0.1, 0.3] and 0.6% (4 instances) higher than 0.3. In contrast, PoFs of BACCHUS distribute rather sparsely, with

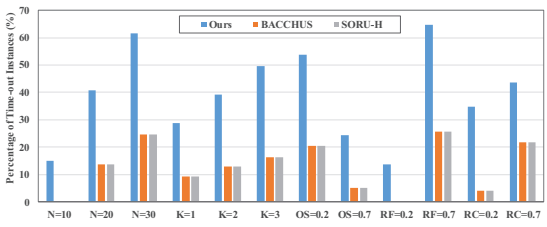

Figure 5: Percentage of time-out instances

28% PoFs within [0.1, 0.3], 5% higher than 0.3, and nearly 67% smaller than 0.1. In addition, most of the instances with OS=0.7 have PoFs smaller than 0.1. These results indicate that the summarization heuristic tends to *over-compensate* for α, i.e. produces α-RM that is higher than required, especially for instances with higher OS. Since our algorithm solves SAA problems with tens to hundreds of samples instead of a representative one, better estimation and control of the risk level can be achieved.

Finally, we briefly report the computational efficiency. Note that while our algorithm solves Problem (6) with hundreds of samples, SORU-H and BACCHUS essentially solve a much simpler deterministic RCPSP since only one summarized sample is used. In general, our algorithm finds the optimal solutions of Problem (6) for nearly 60% of all instances, while SORU-H and BACCHUS optimally solve over 90% of the corresponding deterministic instances. However, it is common in the experiments that a sub-optimal solution given by our algorithm is much better than the optimal solutions given by SORU-H and BACCCHUS. We plot the percentage of time-out instances with the problem parameters in Figure 5. As shown, the three algorithms share the same trend, i.e. the hardness increases with N, K, RF and RC, but decreases with RF. For BACCHUS and our approach, this is perhaps because higher RF and RC implies more alternatives for generating POS hence a larger search space, while higher OS means the AON network G_P is denser and already contains many edges that should present in a POS hence a smaller search space. The reason for N and K is straightforward, since they decide the size of an instance.

7 Conclusions and Future Work

In this paper, we propose a novel approach for risk-aware project scheduling, by exploiting a mathematically coherent measure CVaR. We design a general branch-and-bound framework with efficient bound computation for combinatorial CVaR minimization, and instantiate it to solve the proactive scheduling problem. Empirical results show that our approach scales well to a large number of samples, and provides better risk control and robust makespan. For future work, firstly we plan to improve the computational efficiency using stronger lower bounds and more effective heuristics; secondly, we will extend our approach to RCPSP/max, which is considered in (Varakantham, Fu, and Lau 2016; Fu, Varakantham, and Lau 2016).

Acknowledgments. This work was supported by the National Research Foundation (NRF) Singapore.

References

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

Beck, J. C., and Wilson, N. 2007. Proactive algorithms for job shop scheduling with probabilistic durations. *Journal of Artificial Intelligence Research* 28(1):183–232.

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

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

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

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

Fu, N.; Varakantham, P.; and Lau, H. C. 2016. Robust partial order schedules for rcpsp/max with durational uncertainty. In *ICAPS*, 124–130.

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

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

Laborie, P. 2005. Complete mcs-based search: application to resource constrained project scheduling. In *IJCAI*, 181–186.

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

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

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

Policella, N.; Smith, S. F.; Cesta, A.; and Oddi, A. 2004. Generating robust schedules through temporal flexibility. In *ICAPS*, 209–218.

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

Song, W.; Kang, D.; Zhang, J.; and Xi, H. 2017. Proactive project scheduling with time-dependent workability uncertainty. In *AAMAS*, 221–229.

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

Varakantham, P.; Fu, N.; and Lau, H. C. 2016. A proactive sampling approach to project scheduling under uncertainty. In *AAAI*, 3195–3201.