2012

Robust Execution Strategy for Scheduling Under Uncertainity

Na FU
Singapore Management University, na.fu.2007@smu.edu.sg

Robust Execution Strategy for Scheduling under Uncertainty

by

FU Na

Submitted to School of Information Systems in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Information Systems

Dissertation Committee:

Hoong Chuin LAU(Supervisor / Chair)
Associate Professor of Information Systems
Singapore Management University

Pradeep Reddy VARAKANTHAM
Assistant Professor of Information Systems
Singapore Management University

Shih-Fen CHENG
Assistant Professor of Information Systems
Singapore Management University

Yun Fong LIM
Assistant Professor of Operations Management
Singapore Management University

Singapore Management University
2012

Abstract

Resource Constrained Project Scheduling Problems with minimum and maximum time lags (RCPSP/max) provides a general model for resource scheduling in many real-world problems (such as manufacturing and construction engineering). Due to its practical importance and generality, providing effective algorithms and scalable solutions for RCPSP/max is a topic of growing research. Traditional methods have addressed deterministic models with all parameters known with certainty. In this thesis, we are concerned with RCPSP/max problems in an uncertain environment where durations of activities are stochastic and resource availabilities are subject to unforeseen breakdowns.

We propose methods for generating robust execution strategy to protect against uncertainty. Given a level of risk prescribed by the planner, we investigate the problem of computing the minimum probability-guaranteed makespan (i.e. Robust Makespan) and propose methods for generating an execution strategy such that when uncertainty is dynamically realized, any schedule instantiated from the execution strategy will be guaranteed to finish within the robust makespan with the given level of confidence (or risk).

To the end, we first introduce and study the properties of a decision rule used to compute the start times of all activities with respect to execution strategies and dynamic realizations of uncertainty. Based on the decision rule, we derive the expression for robust makespan of an execution strategy as a quantitative and exact measurement of robustness and propose methods for generating execution strategy with the best robust makespan. To improve performance of local search, we provide enhancements that exploit temporal dependencies between activities. Our experimental results illustrate that robust local search is able to provide robust execution strategies efficiently. We then apply our methodology in the MRO industry context and provide the planner candidate execution strategy based on his own attitude to risk.

Next, we deal with the additional consideration of resource uncertainty due to resource breakdown and repair. We propose different chaining procedures and compute robust resource allocations by predicting the effect on robustness of generated strategies. We incorporate such information into robust local search and propose methods for generating execution strategies that can better absorb resource and durational uncertainties. Experiments show that our proposed model is capable of addressing uncertainties of both resources and durations and the most robust execution strategy is generated by the chaining procedure that takes into account of both mean and variance values of the effect of resource breakdown.

Finally, we apply our model to the service industry context and add visibility from a risk management perspective to the service provider by computing the robust cost of business processes.

Contents

List of Figures

List of Tables

Acknowledgments

Foremost, I would like to thank my supervisor, Prof. Lau Hoong Chuin, for his ZhiYuZhiEn, for his five-year guidance of my Ph.D research, for his patience, enthusiasm, motivation, and immense knowledge, for his care about my graduate life. His rigorous academic style, piercing insights for good research and humorous personality have a profound impact on my future work and life.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Pradeep Reddy Varakantham, Prof. Cheng Shih-Fen, and Prof. Lim Yun Fong, for their time, insightful suggestions and valuable comments.

I am fortunate to know Prof. Pradeep Reddy Varakantham as my mentor. I appreciate his encouragement and experience sharing in doing research. I have benefited intellectually and mentally from cooperation with him. I am also grateful to Prof. Cheng Shih-Fen for guiding me to well organize literatures, without whom I would not have a smooth start of my research.

I would like to express my appreciation to two other cooperators, Xiaofei and Tan Karway, for contributions to Chapter 4 and Chapter 6 of the thesis, respectively. Also sincere thanks to all my SIS friends. Thanks for the friendship and memories.

My deepest gratitude goes to my beloved parents for their nurture and unconditional love. Likewise, I also appreciate the help of my parents-in-law for taking care of my child during my thesis writing. Finally, I would like to express my heartfelt gratitude to my husband, Zhangxin, for his continuous support and encouragement. His incredible support was critical for me to complete the thesis.

To those who indirectly contributed to the thesis, your kindness and help mean a lot to me. Thank you very much.

Dedication

I would like to dedicate this thesis to my husband, Zhangxin and son, Hanwen, for their love.

Chapter 1

Introduction

1.1 Motivation

Scheduling is an important decision-making process in diverse areas such as manufacturing, logistics and MRO, etc. as it describes guidance and pathway for successful completion of a project. Scheduling, in general, is concerned with deciding how to commit resources over time to perform a variety of activities in order to optimize certain criteria (minimal makespan, minimal cost, etc.). For example in manufacturing, a schedule tells a production facility when to make, with which staff, and on which equipment and thus scheduling results makes a major impact on the productivity and efficiency of the production project.

Most research on scheduling typically address deterministic models with all data parameters known with certainty in advance - an assumption which is hardly satisfied in real-world scheduling environments. In general situations where problem parameters are subject to significant random perturbations due to unexpected external events such as manpower unavailability, weather changes, machine breakdowns, etc., deterministic schedules tend to be brittle and may become infeasible. We quote from the work by Bidot et al. [8] on real-world scheduling:

> In practical applications, we have to plan or schedule with incomplete, imprecise, and/or uncertain data ..... as there is a high chance that such a (deterministic) schedule will not fit the real situation that will arise.

This thesis seeks to bridge the gap between theory and practice in real-world scheduling and deals with scheduling in the stochastic environment.

From the computational perspective, stochasticity adds a great deal of complexity to the underlying deterministic scheduling problem. For example, in the infinite-resource project scheduling problem where processing times have two possible discrete values, the problem of computing the expected makespan (or any point on the cumulative distribution of the optimal makespan), is #P-hard [24, 42]. It has also been shown that for the scheduling problem $1|stoch\ p_j; d_j = d|E[\sum w_j U_j]$, the problem of computing a policy (i.e., execution strategy) maximizing the probability that some job completes exactly at the deadline is PSPACE-hard [14]. [12] considered a one-machine scheduling problem with probabilistic durations, with an objective to capture the likelihood that a schedule yields actual performance no worse than a given target level. This has been shown to be NP-hard even though the underlying deterministic problem can be solved in polynomial time.

There are broadly two approaches for tackling scheduling problems in a stochastic environment. One is to adopt a hybrid of proactive and reactive methods (e.g., [57]) where an initial baseline schedule is computed offline, which is then modified (if required) during execution reactively based on the occurrence of external events. The second approach (e.g., [43]) is to design schedule policies that provide online decision rules such that at time t, the policy decides which task(s) may start and which resource(s) to assign. This thesis adopt the latter approach and focus on the computation of a robust execution strategy.

The concrete problem addressed in the thesis is the Resource Constrained Project Scheduling Problem with minimum and maximum time lags (abbrev. RCPSP/max). It is a strongly NP-hard combinatorial optimization problem; and even the decision problem of determining whether an RCPSP/max instance has a feasible solution is NP-complete [4]. Due to its complexity, local search based techniques [17] have achieved success in providing scalable solutions. Taking a cue from this and the recent advancements in robust optimization, we contextualize and improve upon the robust local search [38] for solving RCPSP/max problems under data uncertainty with a risk management perspective.

As a general class of scheduling problems across a broad spectrum of industries, RCPSP/max covers many well-known specific problems such as Resource Constrained Project Schedul-

ing Problem (RCPSP), Job-shop Scheduling Problem (JSP), etc. There is a growing interest to account for data uncertainty for solving such specific problems [26, 5, 52]. However, little work has been done on RCPSP/max problems under uncertainty in the scheduling research. This thesis deals with the uncertainty aspect, in particular, we aim for building robust schedules for RCPSP/max where resource availabilities are subject to unforeseen breakdowns and durations of activities are random variables.

1.2 Research Questions

The major objective of the thesis is to develop methodologies to build *robust* execution strategies for RCPSP/max in a stochastic environment. When refereing to *robustness*, it is naturally to ask three questions [1]:

(1) What behavior makes the execution strategy robust?

(2) What uncertainties must the execution strategy be robust against?

(3) Quantitatively, exactly how robust is the execution strategy?

Besides these three robustness questions, the main research questions we are interested to address is: given only mild statistical information (such as mean and variance) of the input uncertain data (describing stochastic durations and resource breakdowns), can we efficiently and effectively solve RCPSP/max problems within probability guarantee? In particular, given a level of risk (or conference) prescribed by the planner, how to find an execution strategy and a robust objective value such that when resource and durational uncertainties are dynamically realized, the execution strategy will result in a solution whose value is as good as the robust value with the given level of risk (or conference).

To address these questions, we propose methods for solving RCPSP/max problems with stochastic processing times and unfixed resource availabilities from a risk management perspective. More precisely, we compute an execution strategy called Partial Order Schedule (POS) that minimizes the *robust makespan*. Given a level of risk $0 < \varepsilon \leq 1$, the robust makespan is a value for which the probability of realized makespan for any schedule (derived from POS) does not exceed it is greater than $(1 - \varepsilon)$, over all realizations of uncertainty.

1.3 Contributions and Structure

To the best of our knowledge, this is the first work dealing with variable durations, resource breakdowns, and minimum/maximum time lags simultaneously in scheduling. Specifically, the key contributions we made in this thesis are:

- We proposed Generalized Non-Linear Approximation (GNLA) as decision rule and derived robust makespan expression based on GNLA. We applied a local search methodology for generating robust execution strategy. GNLA-based fitness expression is shown to be able to guide local search to generate more robust execution strategies than existing method.

- We provided enhancements to local search by exploring precedence relationship between activities and improving resource chaining based on predicted robustness of execution strategies. Our results demonstrate performance improvement of local search.

- Besides RCPSP/max, we experimented on JSP problems with durational uncertainty and computed the minimum robust makespan with proven probability of successful execution. We also applied the proposed model in the MRO industry to provide the planner guidance on execution strategies.

- We developed a mechanism that translates information of resource breakdowns to (further) duration variability of activities so that our model is able to build execution strategies that hedge against both resource and durational uncertainties. To improve robustness, we proposed different resource chaining procedures to obtain execution strategies based on resource breakdown and repair distributions.

- We applied our model in the service industry context to improve service delivery by providing the robust cost of business processes.

The rest of this thesis is organized as follows: Chapter 2 first reviews the existing studies on RCPSP/max followed by the work on scheduling under uncertainty from four perspectives – Generation of Generic Schedule, Generation of Flexible Schedule, Scenario-based Optimization in Scheduling and Robust Optimization in Scheduling. We provide background and pre-

liminary in Chapter 3. Chapter 4 introduces our first work [20, 22] for RCPSP/max with durational uncertainty, where Generalized Non-Linear Approximation (GNLA) and local search enhancements are proposed and examined. To generate robust execution strategy under both resource and durational uncertainties, we extend the work in [38] and evaluate the effect of resource breakdown in Chapter 5. We then propose different chaining procedures by predicting the effect on robustness. Finally, we apply our proposed model in a service industry context in Chapter 6. Chapter 7 concludes the contribution of this thesis and describes possible directions for future research.

Chapter 2

Literature Review

2.1 Overview

The Resource-Constrained Project Scheduling Problem with minimum and maximum time lags, RCPSP/max, (or known as the Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations, RCPSP-GSR) is a strongly NP-hard combinatorial optimization problem [4]. Compared with its specific problems like RCPSP, JSP, etc. not much attention have been paid to RCPSP/max in the scheduling research. Branch and bound based methods [18] and heuristic approaches [9, 53] have been shown success in solving these problems. For a survey of recent developments and new applications of RCPSP/max, one may refer to [46].

However, to the best of our knowledge, little study has considered RCPSP/max when uncertainty is an issue. One such paper dealing with variable durations on RCPSP/max is done by [41], where activity durations range between given lower and upper bounds. A precedence constraint posting approach [48] was adopted. Whereas in our work, we consider RCPSP/max with durational uncertainty where each activity duration is modeled as a random variable with known mean and variance values. Besides the durational perturbation, we also address resource uncertainty where resource availabilities are subject to unforseen breakdowns. Another work [38] solved RCPSP/max with durational uncertainty by combining techniques from robust optimization with classical local search to compute an execution strategy that minimizes the robust

makespan. The key idea there was to apply the segregated linear decision rule [10] to represent start times of activities and then compute an upper bound for the robust makespan, which is in turn used for guiding the local search to find the robust execution strategy. The use of the segregated linear decision rule provides for computing the fitness measure quickly. However, this advantage comes at a price in that the fitness measure is a loose upper bound (which has the side effect of guiding the search into bad regions). In our work, we address the issue of loose upper bounds by proposing a new general non-linear decision rule and deriving the fitness measure of an execution strategy irrespective of the durational uncertainty distributions.

From the perspective of methodology, research on scheduling with uncertainty has received a growing interest in Artificial Intelligence communities [26, 2, 5, 52, 41, 8]. Broadly, one may classify the techniques to tackle scheduling with uncertainty into two categories: *Proactive Scheduling* is to design a priori schedule or a schedule policy that take into account the possible uncertainty that may occur; *Reactive Scheduling* modifies or re-optimizes the baseline schedule when an unexpected event occurs. This thesis focuses on proactive scheduling and we are concerned with robust scheduling which aims for obtaining robust execution strategies that maintain a high level of performance against uncertainty.

The main idea of *proactive* techniques is to build a global solution which hopefully does not need to be revised at execution time. We first review two class of proactive techniques related with our work, according to how and when the information of uncertainties can be taken into account in generating more robust and stable schedules than they would be without using this information [8]: 1. generating one complete *generic* schedule which is proved to execute correctly in most scenarios arising during execution; 2. generating a *flexible* solution in which some decisions are postponed to be made until execution. After that, we will review the optimization techniques in OR community in solving scheduling problems.

2.2 Generation of Generic Schedule

A first method for making a *generic* schedule that is insensitive to online perturbations is to produce a complete and robust schedule by taking into account all possible scenarios, i.e. a schedule with *strong controllability* [56]. Rather than dealing with execution with 100% con-

fidence, *probabilistic techniques* have been proposed that build schedules with a probabilistic guarantee against a threshold value of an optimization metric such as makespan. Another example of such generic schedule generation is *fuzzy scheduling* [26]: instead of stochastic variables and probabilistic distributions, fuzzy set scheduling use fuzzy numbers for modeling uncertainties based on possibility theory; a recent work by [52] modeled uncertain durations as fuzzy numbers and improved local search to solve the Job Shop Scheduling Problem. In the following, we provide further details on the work related to strong controllability and probabilistic techniques.

2.2.1 Strongly Controllable Techniques

Strong Controllability was introduced by [56] over Simple Temporal Networks with Uncertainty (STNU) for which controllability is achievable in polynomial time. With the existence of uncontrollable events that are controlled by exogenous factors, often referred to as Nature, an STNU is strongly controllable if there exists at least one universal schedule that suits any situation. Such schedule might be computed off-line beforehand. Strong controllability is the strictest form of STNU. A strongly controllable network means that the schedule can be executed without regard to the contingent events. It is useful in applications where contingent events cannot be observed exactly.

2.2.2 Probabilistic Techniques

Instead of generating a global solution suitable for all realizations of uncertainties, probabilistic techniques build a schedule that has a probabilistic guarantee of a deterministic optimization measure with respect to a threshold value, e.g., find the schedule with the highest probability that the project makespan will not exceed a particular value.

[12] defined a β-robust schedule as one that has maximum probability of achieving a given performance level, e.g., the total flow time is no greater than a given threshold. They presented branch-and-bound and heuristic techniques to find a robust schedule in a one-machine manufacturing context that performs the best within a given confidence level. A recent approach [5] provides techniques to compute the robust baseline schedule from a risk management perspec-

tive, where durations of activities are modeled as random variables. Given a value $0 < \alpha \leq 1$, they were interested to compute a schedule with minimal (probabilistic) makespan where the probability of successful execution is at least $1 - \alpha$ over all realizations of the durational uncertainty. The main contribution there was to derive a lower bound for the α-makespan of a given schedule by solving a deterministic problem. They considered the Job-shop Scheduling Problem (JSP) that represents a special case of RCPSP/max (which is the problem of interest in this thesis).

2.3 Generation of Flexible Schedule

Another way of producing robust schedule taking into account of uncertainty is to introduce flexibility into the schedule. The idea is that only a subset of decisions are made offline and the rest are postponed to be made online, so that decisions are only made when information becomes more precise and certain [8]. In the following, we discuss three subcategories of works that deal with generating flexible schedules.

2.3.1 Dynamically Controllable Techniques

An STNU is Dynamically Controllable [56] if there exists a solution that can always be instantiated incrementally based on the outcomes of contingent edges in the past. An execution strategy using dynamic controllability is needed to produce an incremental solution based on the subsequent revelation of contingent events. [44] proposed a pseudo-polynomial algorithm to handle dynamic controllability of STNUs based on constraint satisfaction. Techniques were proposed by [59], to optimize the bounds on durations of contingent edges such that the resulting STNU is dynamically controllable.

2.3.2 Redundancy-based Techniques

Redundancy-based scheduling is another proactive technique for scheduling. The idea is to generate a schedule that includes the allocation of extra resources and/or time in the schedule so that these buffers will help absorb the impact of unexpected events without rescheduling dur-

ing execution. [13] proposed techniques for generating robust schedules based on the insertion of temporal slacks to critical activities that are allocated on possibly breakable resources. [55] proposed heuristics and meta-heuristics to allocate time buffers to a given initial schedule for RCPSP and compared the performance of all algorithms with respect to the stability cost based on simulation. [37] solved RCPSP with uncertain resource availabilities. They analytically determined the expected duration increase an activity experiences due to resource breakdowns. Based on this information, they used simulation-based time buffering to protect the schedule from disruptions caused by resource unavailability. This algorithm, however, is still computationally challenging in solving large-scale scheduling problems as compared to local search, and this approach can not handle uncertain activity durations and unexpected machine breakdowns simultaneously. This thesis is motivated by the above limitation. One contribution of our thesis is in devising a computationally efficient approach based on robust local search that deals with both resource and durational uncertainties in RCPSP/max.

2.3.3 Schedule Policy-based Techniques

Even with buffering, baseline schedules may become brittle in face of unpredictable execution dynamics and get invalidated. Instead of baseline schedule, another line of work is to consider design of good schedule policies(e.g. [43]) that provide online decision rules such that at time t, the policy decides which task(s) may start and which resource(s) to assign. [15] applied resource-based policies for RCPSP with uncertain durations where the decision times are determined by the realization of durational uncertainty. The objective of that work is to determine a project execution policy with minimized execution cost. Another example is the notion of Partial Order Schedules (POS) defined by [49] which seeks to retain temporal flexibility whenever the problem constraints allow it and can often absorb unexpected deviation from predictive assumptions. They considered robustness measures such as fluidity and flexibility. Generating POS is another example of such flexible approaches: a subset of sequencing decisions are made offline and the remaining decisions are made online by using a dispatching rule [8]. Different methods of generating POS were compared in terms of the robustness of the resulting schedules in the work of [50]. In this thesis, we apply the concept of POS as the execution policy.

2.4 Scenario-based Optimization in Scheduling

Another line of work that deals with scheduling under uncertainty is based on the use of scenarios (scenario-based optimization). For example, [33] introduced the concept of robustness into scheduling problems. They considered uncertain processing times and proposed methods to generate a robust schedule based on the maximum absolute deviation between the robust solution against all possible scenarios in a given scenario set. A shortcoming of this kind of approach is that all scenarios are assumed to be known in advance, and that the scenario space is usually exponentially large. Noteworthy of mention are the two notions of solution robustness and quality robustness, where solution robustness (or stability) refers to the insensitivity of actual start times, whereas quality robustness refers to the insensitivity of solution quality (i.e. makespan) to different scenarios [26]. Another pioneering scenario-based optimization work is by [45] which handles the tradeoff between solution robustness (if a solution remains close to the optimal for all scenarios) and model robustness (if a solution remains feasible for most scenarios).

2.5 Robust Optimization in Scheduling

A recent development in Operations Research saw the potential of applying the concept of Robust Optimization to deal with uncertainty. [6] and [7] proposed robust optimization models where no assumptions of the underlying probability distribution of data are needed. The idea is often to approximate data uncertainty by a tractable (convex) uncertainty set, and optimization is performed on that set. This results in a robust counterpart formulation as a conic (such as second-order cone) optimization problem which can be solved in polynomial time. However, only a few works have been reported in the literature on applying robust optimization to scheduling, due mainly to a high-degree combinational nature of the problem. One such application is the process scheduling problem in chemical engineering, such as the works by [28] and [40]. A notable recent breakthrough in robust optimization on tractable approximation models to solve stochastic optimization problems is found by [10]. This approach, while tractable, may still be computationally challenging in solving large-scale optimization problems as compared to local search. Motivated by the above limitations, we propose methods

for generating the robust execution strategy with the (locally) minimum robust makespan for RCPSP/max under the situation that activity durations are uncertain and resources are subject to unexpected breakdowns.

Chapter 3

Background

3.1 Definitions of RCPSP/max

3.1.1 Deterministic RCPSP/max

The RCPSP/max problem [4] consists of *N* activities $\{a_1, a_2..., a_N\}$, where each activity a_j $(j = 1, ...N)$ is to be executed for a certain amount of time units without preemption. Each activity a_j has a fixed *duration* or *processing time* d_j, which is assumed to be a non-negative real number or non-negative integer number. In addition, dummy activities a_0 and a_{N+1} with $d_0 = d_{N+1} = 0$ are introduced to represent the beginning and the completion of the project, respectively.

A start time *schedule* ***ss*** is an assignment of start times to all activities $a_1, a_2..., a_N$, i.e. a vector $\boldsymbol{ss} = (st(a_1), st(a_2), ...st(a_N))$ where $st(a_i)$ represents the start time of activity a_i and $st(a_0)$ is assumed to be 0. Let $et(a_i)$ be the end time of activity a_i. Since durations are deterministic and preemption is not allowed, we then have

$$st(a_i) + d_i = et(a_i). \tag{3.1}$$

And the project *makespan* which is also the start time of the final dummy activity $st(a_{N+1})$ equals

$$st(a_{N+1}) = max_{i=1,...N} et(a_i). \tag{3.2}$$

Schedules are subject to two kinds of constraints, *temporal constraints* and *resource constraints*. Temporal constraints restrict the time lags between activities. A *minimum time lag* T_{ij}^{min} between the start time of two different activities a_i and a_j says that

$$st(a_j) - st(a_i) \geq T_{ij}^{min} \tag{3.3}$$

Specially, $T_{ij}^{min} = 0$ means that activity a_j cannot be started before activity a_i begins. A *maximum time lag* T_{ij}^{max} between the start time of two different activities a_i and a_j says that

$$st(a_j) - st(a_i) \leq T_{ij}^{max} \tag{3.4}$$

$T_{ij}^{max} = 0$ means that activity a_j cannot be started after activity a_i begins.

In this definition, time lags connect start times of two related activities, known as *start-to-start* time lags. *start-to-end*, *end-to-end*, *end-to-start* time lags can be easily transformed to the general *start-to-start* time lags for the deterministic case as given by [4]. A schedule $\boldsymbol{ss} = (st(a_1), st(a_2), ...st(a_N))$ is *time feasible*, if all the time lag constraints are satisfied at the start times $st(a_i)$ $(i = 1, ...N)$.

A resource unit is reusable and available for another activity once it is no longer used by the current activity. Each type of resource has a limited capacity, C_k $(k = 1, 2..., K)$ units. Each activity a_i requires r_{ik} units of resource of type k where $k = 1, 2..., K$. Let $A(t) = \{i \in \{1, 2...N\} | st(a_i) \leq t \leq et(a_i)\}$ be the set of activities which are being processed at time instant t. A schedule is *resource feasible* if at each time instant *t*, the total demand for a resource k does not exceed its capacity C_k, i.e.

$$\sum_{i \in A(t)} r_{ik} \leq C_k. \tag{3.5}$$

A schedule ***ss*** is called *feasible* if it is both time and resource feasible. The objective of the deterministic RCPSP/max scheduling problem is to find a feasible schedule so that the project makespan is minimized.

Example 1. *In Figure 3.1, we show a simple example of a deterministic RCPSP problem which is a special case of RCPSP/max with only precedence constraints (rather than arbitrary time*

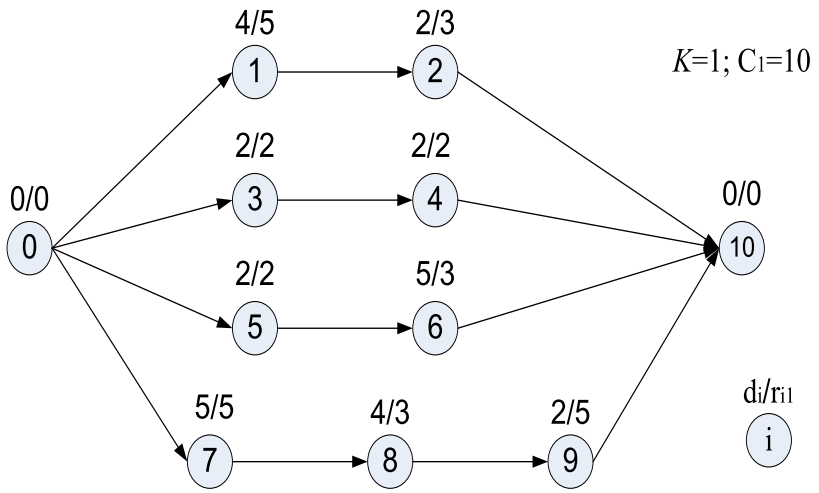


Figure 3.1: Project Instance.

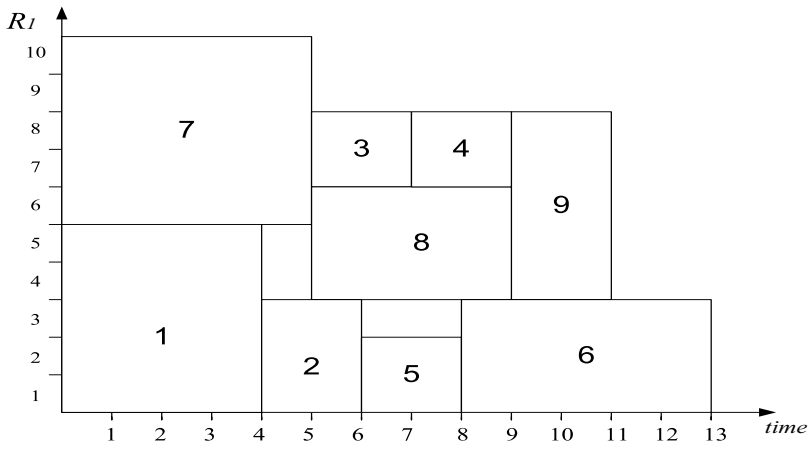


Figure 3.2: Example Schedule.

lags) between activities for expository purposes. Each circle indicates an activity with the number inside the circle representing the activity ID. The two numbers on top of each activity represent the duration and the number of units of the resource required by the activity. In this example, there are 9 activities and one type of resource, with the capacity of the resource limited to 10. It should be noted that the activities 0 and 10 are dummy activities introduced to have a source and sink in the dependency graph. Arrows between activities represent temporal dependencies. A feasible schedule with makespan of 13 is represented in Figure 3.2.

3.1.2 Data Uncertainty and Robust Makespan

The duration of an activity is specified as a sum of its mean value and its deviation: $\tilde{d_i} = d_i^0 + \tilde{z}_i$, where d_i^0 is the mean of $\tilde{d_i}$ and $\tilde{z}_i$ is the perturbation part with an expected value of 0 and standard deviation σ. It should be noted that irrespective of its distribution type, we can always represent $\tilde{d_i}$ as $\tilde{d_i} = d_i^0 + \tilde{z}_i$ where d_i^0 is the mean and $\tilde{z}_i$ is the perturbation part with $E(\tilde{z}_i) = 0$. In addition, we also assume that these random variables, $\{\tilde{z}_i\}$, corresponding to durational uncertainty are independent of each other.

Similar to the deterministic RCPSP/max, the *start-to-start* constraints are still deterministic. However, unlike the deterministic case, other types of constraints (*end-to-start* etc.) cannot be converted into deterministic *start-to-start* constraints . Instead the equivalent *start-to-start* constraint is a stochastic one as shown in the following expressions for an *end-to-start* constraint. It should be noted that even though the converted constraints are stochastic, our techniques will still be applicable (with minor modifications) to all types of time lag constraints. Our robust local search techniques depend on the computation of maximum and sum of random variables and even with stochastic time lag constraints that remains the case. In this work, for purposes of exposition, we present our techniques assuming the temporal dependencies are provided as *start-to-start* constraints.

$$
\begin{aligned}
st(a_j) - et(a_i) &\leq T_{ij}^{max} \\
st(a_j) - (st(a_i) + \tilde{d_i}) &\leq T_{ij}^{max} \\
st(a_j) - st(a_i) &\leq T_{ij}^{max} + \tilde{d_i}
\end{aligned}
$$

In the deterministic setting, start time schedules can be computed and values of makespan can be used to evaluate the performance of the schedule. However, when durational uncertainty is involved, the project makespan becomes a random variable and the schedule is replaced by an execution strategy. In the following sections, we introduce the *Partial Order Schedule* (POS) [49], which serves as an execution strategy of the scheduling project.

Given a level of risk $0 < \varepsilon \leq 1$, the goal of our problem is to find such a strategy with a minimum value (across all strategies) of the robust makespan. We define the $robust\ makespan$ as a makespan value where the probability that any feasible schedule (i.e. an assignment of start times to activities) instantiated from the strategy can be completed before robust makespan is at least $1 - \varepsilon$.

3.1.3 An Instance: Job-shop Scheduling Problem (JSP)

Due to the generality of RCPSP/max, this model covers many difficult special problems that has been studied in literature, like Job-shop Scheduling Problem(JSP). The classical JSP consists of a set of n jobs and a set of M machines. Each job J_i ($i = 1, ...n$) consists of a sequence of n_i operations denoted as O_{ij} ($j = 1, ...n_i$) which have to be processed in a given order. For convenience, we enumerate all operations of all jobs by O_k, where $k = 1, ...N$ and $N = \sum_{j=1}^{n} n_j$. Each operation O_k has a positive duration denoted as d_k and must be executed on a dedicated machine denoted as M_k. Once an operation is started it must be executed for its entire duration. No operations that require the same resource can overlap in their execution. Thus, operations can be partitioned into two sets: job sets and resource sets. Job sets referring to operations corresponding to a job and resource sets referring to all operations that require the same resource.

A solution *s* is a total ordering of operations on each resource set, which does not conflict with the job ordering. A *path* of a solution *s* is a sequence of operations which follows both the job ordering and the ordering on various resource sets of the solution *s*. The *length* of a path is equal to the sum of the durations of the operations in the path. The *makespan* of a solution *s make(s)* is the length of the longest path. *The minimum makespan* of a JSP problem is defined to be the minimum value of makespans over all solutions, i.e. $min_s\ make(s)$. Each

operation O_k is associated with a start time of $st(O_k)$ and end time of $et(O_k)$. A *schedule* is an assignment of starting times $st(O_k)$ ($k = 1, ...N$) to all operations on the machines. The objective is to find a schedule which optimizes the total makespan (makespan is the completion time of the last operation): $max_{k=1}^{N} et(O_k)$, which is also the minimum value of the longest path of all solutions. The job shop scheduling problem is a special case of RCPSP in which resources have unary capacity and each activity (i.e. operation) consumes only one resource.

We can propagate the same notations from RCPSP/max with durational uncertainty to the JSP with durational uncertainty, i.e. the processing time of each activity (i.e. operation) $\tilde{d}_{O_k}$ is now modeled as a sum of an expected value $d^0_{O_k}$ and a random part $\tilde{z}_{O_k}$: $\tilde{d}_{O_k} = d^0_{O_k} + \tilde{z}_{O_k}$. The objective is to find the robust makespan with a given level of risk.

3.2 Execution Strategy: Partial Order Schedule (POS)

A *Partial Order Schedule* (POS) was first proposed by [49]. It is defined as a set of activities, which are partially ordered such that any schedule with total activity order that is consistent with the partial order is resource and time feasible. Mathematically, a POS can be represented by a graph where a node represents an activity and the edges represent the precedence constraints between the activities. Within a POS, each activity retains a set of feasible start times, which provide the flexibility to respond to unexpected disruptions. A POS can be constructed from a given RCPSP instance via a chaining algorithm (where one such algorithm is described below).

Example 2. *Figure 3.3 provides a POS for the problem instance introduced in Example 1. There are 10 units of the resource and that is shown on the left most side of the figure. Each unit represents a chain. An activity can require multiple resource units and hence is shown on multiple resource units. For instance, activity 6 is shown on resource units 4, 7 and 8. A solid arrow between activities represents a temporal dependency provided in the original problem. Solid arrow between activities 1 and 2 is one such example. A dotted arrow between activities represents a temporal dependency that is introduced since both activities have to be executed on the same resource unit. It is added to remove resource conflict. An example for this is the dependency introduced between activity 2 and activity 6. For explanatory purposes we*

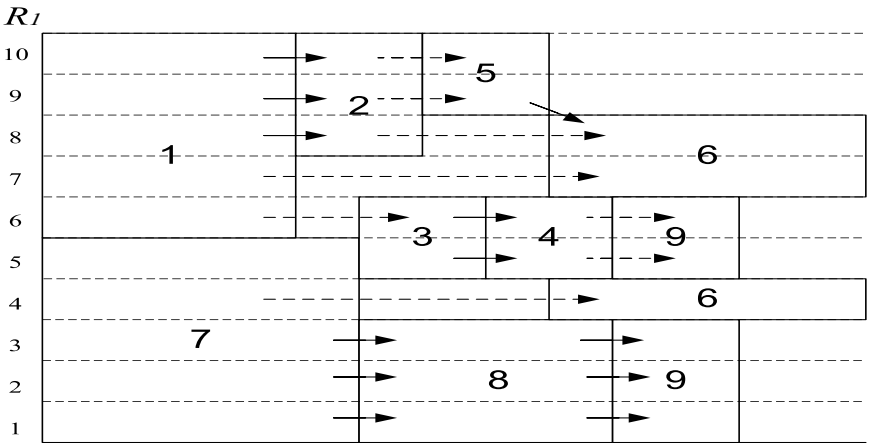


Figure 3.3: Example of POS

only consider one resource type in this example, however in the most general case, there exists multiple resource types and a dependency diagram for every resource type.

3.2.1 Chaining Algorithms

Chaining is a procedure of dispatching activities to different resource units (henceforth referred to as chains) based on temporal and resource feasibility. During the chaining process, each activity can be allocated to one or more resource chains based on the number of resource requirement of the activity. During the chaining process, once an activity is scheduled to be executed on a resource unit, an additional edge (indicating precedence relationship) is added between the last activity of the selected chain and this activity so as to eliminate all possible resource conflicts.

In the following, we describe the basic chaining algorithm proposed by [49]. In this algorithm, a feasible schedule is first obtained using a simple greedy heuristic. Consequently, the POS is constructed through a chaining method as follows: First, the set of activities are sorted according to their start times given in the feasible solution; Then, all activities are allocated to different chains in that order, where each chain corresponds to a unit of a certain type of resource. A chain is called *available* for an activity if the end time of the last activity allocated on this chain is no greater than the start time of the activity in the feasible schedule. Once an activity is allocated on a chain, a precedence constraint between this activity and the last activity of the chain is posted. For those activities that require more than one unit of one or

more types of resources, they will be allocated to a number of chains with the number equal to the overall number of resource units required by the activity.

Example 3. *Take Figure 3.3 for example. Given the schedule of Figure 3.2 as an input,activities are first sorted according to their starting time and the sequence of activities can be presented as:* (*7,1,2,8,3,5,4,6,9*). *The chaining procedure first picks activity 7 and randomly allocates it to five chains to fulfill its resource requirement. The available chains are those belonging to dummy activity 0.Thus, five chains 1 through 5 are created which posts the precedence relationship from the current last activity 0 to activity 7. Activity 7 then becomes the last activity on those chains. Activity 1 is treated in the same way. The available chains for activity 2 are those belonging to activity 1. Activity 2 is then randomly assigned to chain 8 through 10 and an edge between activity 1 and activity 2 indicating precedence relationship is added. This procedure continues until all activities are dispatched to chains that the number equals its resource requirement, and finally the chained POS 3.3 is yielded. However, because the randomness of the chaining procedure, activity 6 is allocated to chains that belong to three different activities: activity 2, activity 1 and activity 7. This will tie together the execution of three previously unrelated activities: (activity 2, activity 6),(activity 1, activity 6) and (activity 7, activity 6), which would decrease the flexibility of execution.*

To reduce inter-dependencies between activities as much as possible during the chaining procedure, [47] developed two heuristics. One direct advantage of such approaches is that synchronization points of a solution can be reduced:

- Activities that require more than one resource units are allocated to the same subset of chains. This is achieved by scanning the list of available chains where the last activity in the chain : (a) requires multiple resource units; and (b) was also previously assigned another resource unit allocated to the current activity.
- Activities with a precedence constraint defined in the original problem are allocated to the same set of chains. This is implemented by choosing a chain that has a last activity with precedence constraint with the current activity.

Example 4. *Figure 3.4 provides the POS computed by using the above mentioned chaining algorithm for the RCPSP problem described in Example 1. When allocating activity 6, the*

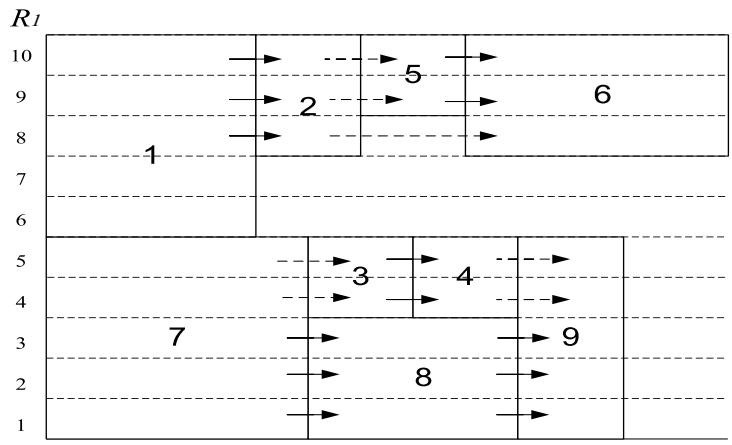


Figure 3.4: POS computed with Removed Synchronization Point

available chains are divided into two sets: {*chain 10, chain 9*} *and* {*chain 8, chain7, chain6*}. *The first set contains chains for which the last activity (i.e. activity 5) is already ordered in problem definition with respect to activity 6. A chain (for example, chain 10) is randomly chosen from this set with the last activity on it as activity 5. Then, The remaining available chains for activity 6 is redivided into two sets:* {*chain 9*} *and* {*chain 8, chain7, chain6*}. *The first set contains the chains with activity 5 (i.e. the last activity of the first picked chain) as the last activity and the second set are the remaining. Activity 6 is first allocated to chains belonging to the first subset to satisfy all remaining resource requirements. In this case, the synchronization points caused by activities 1 and 6, activities 7 and 6 being allocated to different chains has disappeared.*

3.3 Segregated Random Variables

As given by [6], we also classify the variables in a stochastic optimization problem into 2 types: *Adjustable* and *Non-Adjustable variables*.

Definition 1. *Non-Adjustable variables are a priori decisions that must be made before the actual realization of the uncertainty.*

Definition 2. *Adjustable variables (also known as recourse variables) are 'wait-and-see' variables that can adjust themselves when part of the uncertain data become known.*

For example, in a scheduling problem such as RCPSP with uncertain task durations, the non-adjustable variables will represent the execution policy, e.g., the POS proposed by [49], that need to be constructed a priori, while the adjustable variables are associated with the actual start times of the tasks, which will be set with respect to the execution policy and dynamic realizations of uncertainty.

A random variable will be denoted by $\tilde{x}$ and bold face lower case letters such as $\mathbf{x}$ represent vectors.

A *primitive* random variable $\tilde{z}_k$ is one which has zero mean. Examples of a primitive random variable include $U(-a, a)$ (uniform distribution between constants $-a$ and a) and $N(0, \sigma)$ (normal distribution with mean 0 and variance σ^2). As mentioned earlier, we assume that every uncertain distribution is equal to the sum of its nominal value (mean) and its deviation, represented by one (or possibly more) primitive random variable $\tilde{z}$. In a straight forward representation, there is only one primitive random variable $\tilde{z}_k$ associated with an uncertain variable. In the recent work by [10], each primitive random variable $\tilde{z}_k$ is represented by 2 *segregated* random variables $\tilde{z}_k^+$ (read z-plus) and $\tilde{z}_k^-$ (z-minus):

$$\begin{aligned} \tilde{z} &= \tilde{z}^+ - \tilde{z}^- \\ \tilde{z}^+ &= max\left\{\tilde{z}, 0\right\} \\ \tilde{z}^- &= max\left\{-\tilde{z}, 0\right\}. \end{aligned} \tag{3.6}$$

In the following Table 3.1, we give examples of the respective values of mean μ_p, μ_m and variance ${\sigma_p}^2$, ${\sigma_m}^2$ for the segregated variables $\tilde{z}^+$ and $\tilde{z}^-$.

$\tilde{z}$	$Var(\tilde{z})$	${\sigma_p}^2$, ${\sigma_m}^2$	μ_p,μ_m
$U(-a, a)$	$\frac{a^2}{3}$	$\frac{5a^2}{48}$	$\frac{a}{4}$
$N(0, \sigma)$	σ^2	$\frac{(\pi-1)\sigma^2}{2\pi}$	$\frac{\sigma}{\sqrt{2\pi}}$

Table 3.1: Values of the mean and variance for the segregated variables under Uniform and Normal Distribution

The underlying assumption with the use of segregated random variables is that the mean and variance of the individual segregated variables is provided for the random variables employed. We are not aware of mean and variance values for segregated variables for distributions other than normal and uniform.

3.4 Decision Rules

3.4.1 Decision Rules for Optimization under Data Uncertainty

In optimization problems with data uncertainty, a decision rule specifies the dependence of adjustable variables on the uncertainty parameters and the non-adjustable variables. Let $\tilde{\mathbf{z}}$ and $\mathbf{x}$ denote the set of primitive random variables and non-adjustable variables respectively. An example is the linear decision rule proposed by [6], where the setting value of an adjustable decision variable $\tilde{S}(\mathbf{x}, \tilde{\mathbf{z}})$ is assumed to be affinely dependent on a subset of the N number of primitive random variables:

$$\tilde{S}(\mathbf{x}, \tilde{\mathbf{z}}) = c^0 + \sum_{k=1}^{N} c_k(\mathbf{x})\tilde{z}_k \tag{3.7}$$

where each $c_k(\mathbf{x})$ $(1 \leq k \leq N)$ is a coefficient derived from $\mathbf{x}$.

Another example is the segregated linear decision rule proposed by [10], where each adjustable decision variable is assumed to be affinely dependent on a set of some N segregated random variables $\left\{\tilde{z}_1^+, \tilde{z}_1^-, \ldots, \tilde{z}_N^+, \tilde{z}_N^-\right\}$. Hence, a segregated linear decision rule has the following general form:

$$\tilde{S}(\mathbf{x}, \tilde{\mathbf{z}}) = c^0 + \textstyle\sum_{k=1}^{N} \left\{c_k^+ \tilde{z}_k^+ + c_k^- \tilde{z}_k^-\right\}. \tag{3.8}$$

As we will show below, a segregated linear decision rule allows us to easily obtain an upper bound on a subset of random variables (see Eqn 3.16), which is not possible in the linear decision rule proposed by [6].

Given the mean and variance for each segregated variable $E(\tilde{z}_k^+) = E(\tilde{z}_k^-) = \mu_k$, $Var(\tilde{z}_k^+) = \sigma_{pk}^2$ and $Var(\tilde{z}_k^-) = \sigma_{mk}^2$, we can express the expected value and variance of any adjustable variable as:

$$E[\tilde{S}(\mathbf{x}, \tilde{\mathbf{z}})] = c^0 + \textstyle\sum_{k=1}^{N} \left\{c_k^+ \mu_k + c_k^- \mu_k\right\} \tag{3.9}$$

$$Var[\tilde{S}(\mathbf{x}, \tilde{\mathbf{z}})] = \textstyle\sum_{k=1}^{N} \left\{ \left[c_k^+ \sigma_{pk}\right]^2 + \left[c_k^- \sigma_{mk}\right]^2 - 2c_k^+ c_k^- \mu_k \right\}. \tag{3.10}$$

3.4.2 Segregated Linear Approximation (SLA) in Scheduling

In this decision rule, the duration for each activity is defined based on the segregated random variables introduced in Section 3.3. For an uncertain duration $\tilde{d}$ with mean processing time d^0, we represent $\tilde{d}$ as a sum of three components: its mean d^0, lateness $\tilde{z}^+$ (i.e. $max\{\tilde{d} - d^0, 0\}$), and earliness $\tilde{z}^-$ (i.e. $max\{d^0 - \tilde{d}, 0\}$),

$$\tilde{d} = d^0 + \tilde{z}^+ - \tilde{z}^-. \tag{3.11}$$

For a normally distributed duration, i.e., $\tilde{z} \sim N\{0, \sigma\}$, the respective values of mean and variance for the segregated variables can be summarized as:

$$E[\tilde{z}^+] = E[\tilde{z}^-] = \frac{\sigma}{\sqrt{2\pi}} \qquad Var[\tilde{z}^+] = Var[\tilde{z}^-] = \frac{(\pi-1)\sigma^2}{2\pi}. \tag{3.12}$$

Now we describe the computation of $\tilde{S}_v(\mathbf{x}, \tilde{\mathbf{z}})$ by representing durational uncertainty for activities using segregated random variables. Upper bounds on both the sum and maximum of random variables are derived as linear functions of segregated variables as illustrated below:

- *Sum* of random variables : In the case of a project network involving k activities, any two of which have either precedence constraints in between or competing for the same resource units, a solution in the form of POS requires computation of the *sum* of activity durations. The start time of the activity starting after the k-activity project is expressed as:

  $$\tilde{S}_k(\mathbf{x}, (\tilde{\mathbf{z}}^+, \tilde{\mathbf{z}}^-)) = \textstyle\sum_{i=1}^{k}(d_i^0 + \tilde{z}_i^+ - \tilde{z}_i^-). \tag{3.13}$$

  Thus, the adjustable variable $\tilde{S}_k$ a mean of $\sum_{i=1}^{k} d_i^0$ with uncertainty captured by a random variable, which has a positive segregated component of $\sum_{i=1}^{k} \tilde{z}_i^+$ and a negative segregated component of $\sum_{i=1}^{k} \tilde{z}_i^-$. Mean and variance of the segregated variables are known and hence the mean and variance of $\tilde{S}_k$ are easy to compute.

- *Max* of random variables: Consider activities that are executed concurrently, the upper bound on the start time of an activity starting after the parallel k-activity project network in SLA is represented by a linear function of the positive segregated components of

duration perturbations:

$$\tilde{S}_k(\mathbf{x}, (\tilde{\mathbf{z}}^+, \tilde{\mathbf{z}}^-)) \leq \max_{i=1,\ldots k}\{d_i^0\} + \sum_{i=1}^{k} \tilde{z}_i^+. \tag{3.14}$$

Thus, the adjustable variable $\tilde{S}_k$ has an upper bound on the mean of $\max_{i=1,\ldots k}\{d_i^0\}$ with uncertainty captured by a random variable with the positive segregated component given by $\sum_{i=1}^{k} \tilde{z}_i^+$ and no negative segregated component. Mean and variance of the segregated variables are known and hence the mean and variance of $\tilde{S}_k$ are easy to compute.

Since, in both cases (*sum* and *max*) $\tilde{S}_k$ is expressed linearly on a subset of random segregated variables, the recursive computation is straightforward. Compared with other linear decision rules [6], the superiority of SLA [10] lies in this ability to linearly express an upper bound on a subset of random variables by dissecting each uncertainty into its positive and negative components. While this approximation increases tractability and scalability, it comes at the expense of losing accuracy.

In RCPSP/max with durational uncertainty, a decision rule specifies the dependence of activity start times on the durational uncertainty associated with other activities. To make the comparison with Equation 3.7, $\mathbf{x}$ represents the POS to be generated; each task's start time is associated with the adjustable variable $\tilde{S}(\mathbf{x}, \tilde{\mathbf{z}})$, where c^0 represents the earliest start time of this task under the POS, and $c_k(\mathbf{x})$ encodes how task k is related to this task in the POS.

In a scheduling context, the start time of an activity is dependent on the start times of the preceding activities, i.e. Adjustable variables $\tilde{S}(\mathbf{x}, \tilde{\mathbf{z}})$ are dependent on one another. Any activity will either start after the end of an activity (i.e. in series) or after the end of multiple activities occurring simultaneously (i.e. in parallel). Thus, adjustable variables are functions of other adjustable variables through the addition operator (to model serial activities) and/or the maximum operator (to model parallel activities).

Given M number of adjustable variables, we may express its sum as an adjustable variable in the form of a segregated linear decision rule as follows:

$$\sum_{i=1}^{M} \tilde{S}_i(\mathbf{x}, \tilde{\mathbf{z}}) = \sum_{i=1}^{M} c_i^0 + \sum_{k=1}^{N} \left\{ \sum_{i=1}^{M} c_{i,k}^+ \tilde{z}_k^+ + \sum_{i=1}^{M} c_{i,k}^- \tilde{z}_k^- \right\}. \tag{3.15}$$

Similarly, given some set C of adjustable variables, we may also express the upper bound on the maximum of these variables as an adjustable variable in the form of a segregated linear decision rule:

$$\max_{i\in C}\{\tilde{S}_i(\mathbf{x},\tilde{\mathbf{z}})\} \leq \max_{i\in C}\{c_i^0\} + \sum_{k=1}^{N}\left\{\max_{i\in C}\{c_{i,k}^+\}\tilde{z}_k^+\right\} + \sum_{k=1}^{N}\left\{\max_{i\in C}\{c_{i,k}^-\}\tilde{z}_k^-\right\}. \tag{3.16}$$

3.5 Robust Fitness Function

The makespan (start time of the dummy "sink" activity) for the RCPSP/max with durational uncertainty is a function of non-adjustable variables $\mathbf{x}$ and random variables representing durational uncertainty $\tilde{\mathbf{z}}$ and is represented using $\tilde{S}(\mathbf{x},\tilde{\mathbf{z}})$ for SLA and $\tilde{G}(\mathbf{x},\tilde{\mathbf{z}})$ for GNLA. Recall that the robust optimization problem is to find the minimum value F^* for which the following probability bound is observed[1]:

$$P(\tilde{S}(\mathbf{x},\tilde{\mathbf{z}}) \leq F^*) \geq (1-\epsilon) \tag{3.17}$$

From the one-sided Chebyshev's Inequality, we can obtain a bound for the robust objective value F^* as a function of its expected value and variance of the adjustable fitness function, i.e.:

$$E[\tilde{S}(\mathbf{x},\tilde{\mathbf{z}})] + \sqrt{\tfrac{1-\epsilon}{\epsilon}}\sqrt{Var[\tilde{S}(\mathbf{x},\tilde{\mathbf{z}})]} \leq F^* \Rightarrow P(\tilde{S}(\mathbf{x},\tilde{\mathbf{z}}) \leq F^*) \geq (1-\epsilon) \tag{3.18}$$

Hence, we can reformulate our robust optimization problem as follows:

$$\begin{aligned} \min \quad & F^* \\ s.t. \quad & E[\tilde{S}(\mathbf{x},\tilde{\mathbf{z}})] + \sqrt{\tfrac{1-\epsilon}{\epsilon}}\sqrt{Var[\tilde{S}(\mathbf{x},\tilde{\mathbf{z}})]} \leq F^* \end{aligned} \tag{3.19}$$

From this model, we can now derive the robust fitness function which will be used in local search:

Definition 3. *Given* $0 < \epsilon \leq 1$ *and the adjustable fitness function* $\tilde{S}(\mathbf{x},\tilde{\mathbf{z}})$ *defined above, the*

[1]We show the computation of SLA robust fitness function. By substituting $\tilde{S}$ with $\tilde{G}$, we obtain the fitness function for GNLA.

robust fitness function, $f(x, \tilde{z}, \epsilon)$, is defined as

$$f(\mathbf{x}, \tilde{\mathbf{z}}, \epsilon) = E[\tilde{S}(\mathbf{x}, \tilde{\mathbf{z}})] + \sqrt{\frac{1-\epsilon}{\epsilon}}\sqrt{Var[\tilde{S}(\mathbf{x}, \tilde{\mathbf{z}})]} \tag{3.20}$$

The goal of the local search mechanism is to find a local minima of f. In addition, local search typically requires the fitness function to be computed many times and hence it is imperative that the computation of fitness function is efficient.

Chapter 4

Robust Execution Strategy under Durational Uncertainty

4.1 Contributions and Structure

In this chapter, we provide a scalable method for solving RCPSP/max problems with durational uncertainty. We introduce the robust local search method consisting of three key ideas:

- Introducing and studying the properties of a decision rule: General Non Linear Approximation in Section 4.2.1, used to compute start times of activities with respect to dynamic realizations of durational uncertainty;
- Deriving the expression for *robust makespan* of an execution strategy based on decision rule approximations in Section 3.5;
- A robust local search mechanism to efficiently compute activity execution strategies that are robust against durational uncertainty in Section 4.3.

Furthermore, we also provide enhancements in Section 4.4 to local search that exploit temporal dependencies between activities.

Experimental results are evaluated in Section 4.5, followed by an application in the MRO industry in Section 4.6 and the summation in Section 4.7.

4.2 Decision Rule for RCPSP/max with Durational Uncertainty

In a scheduling context, the start time of an activity is dependent on the start times of the preceding activities, i.e. Adjustable variables $\tilde{S}(\mathbf{x}, \tilde{\mathbf{z}})$ are dependent on one another. Any activity will either start after the end of an activity (i.e. in series) or after the end of multiple activities occurring simultaneously (i.e. in parallel). Thus, adjustable variables are functions of other adjustable variables through the addition operator (to model serial activities) and/or the maximum operator (to model parallel activities).

More specifically, the output of solving a RCPSP/max involves a POS that is represented as a graph with activities as vertices and precedence constraints between activities as the edges. Given a POS graph, $\mathbf{x} = (V, E)$, where V is the set of activities and E is the set of temporal dependencies (an edge (u, v) represents a temporal dependency that states that activity v should occur after activity u). For any activity $v \in V$, the decision rule for computing its start time is defined recursively as follows:

$$\tilde{S}_v(\mathbf{x}, \tilde{\mathbf{z}}) = \max_{(u,v)\in E}\{d_u^0 + \tilde{z}_u + \tilde{S}_u(\mathbf{x}, \tilde{\mathbf{z}})\}. \tag{4.1}$$

Equation 4.1 is a recursive expression that is defined as a combination of *sum* and *maximum* on a set of random variables. It should be noted that combinations of *sum* and *maximum* of random variables cannot be computed exactly and hence *operational* decision rule approximations: Segregated Linear Approximation(SLA) in Section 3.4.2 and General Non-Linear Approximation(GNLA) in Section 4.2.1 to evaluate the recursive expression of Equation 4.1 are presented.

4.2.1 General Non Linear Approximation (GNLA)

While SLA is efficient, it can typically provide loose upper bounds on robust makespan due to the linear approximation for computing $\max$ of random variables. In this section, we describe General Non Linear Approximation (GNLA), which is not restricted to only affine dependencies. For clarity and comparison purposes, we use $\tilde{G}$ to denote the start time instead of $\tilde{S}$ used

in SLA.

Given the mean and variance values of duration uncertainty, we describe the approximation involved in computing mean and variance of the *sum* and *max* of activities that will be used in Equation 4.1. It should be recalled that irrespective of the distribution of the uncertain duration $\tilde{d}$, we can always represent $\tilde{d}$ as $\tilde{d} = d^0 + \tilde{z}$, where d^0 is the mean of $\tilde{d}$ and $\tilde{z}$ is the perturbation part. Thus, $E(\tilde{z}) = 0$.

Sum of Random Variables

We compute sum of all stochastic durations in a serial k activity project network as follows:

$$\tilde{G}_k(\mathbf{x}, \tilde{\mathbf{z}}) = \sum_{i=1}^{k} (d_i^0 + \tilde{z}_i). \tag{4.2}$$

In this case, we have a similar representation to SLA. Mean and variance of $\tilde{G}_k$ are computed as follows:

Since $\{\tilde{z}_i\}_{i=1,...k}$ are random variables with zero mean, we can then calculate the expected value as:

$$E[\sum_{i=1}^{k} (d_i^0 + \tilde{z}_i)] = \sum_{i=1}^{k} d_i^0. \tag{4.3}$$

Because $\{\tilde{z}_i\}$ are assumed to be independent of each other, the variance value is computed by the following expression:

$$Var[\sum_{i=1}^{k} (d_i^0 + \tilde{z}_i)] = \sum_{i=1}^{k} Var[\tilde{z}_i], \tag{4.4}$$

and under normal distribution where $\tilde{z}_i \sim N(0, \sigma_i)$, we have

$$Var[\sum_{i=1}^{k} (d_i^0 + \tilde{z}_i)] = \sum_{i=1}^{k} \sigma_i^2. \tag{4.5}$$

Note that the expressions for expected value and variance in the case of serial activities are identical to the ones used by [61].

Max of Random Variables

For ease of explanation, we begin by considering two activities to be executed in parallel and then extend the analysis to multiple parallel activities. In GNLA, (unlike in SLA) the *max* of random variables itself is not approximated but the expected value and variance of the *max* are approximately calculated.

Expected Value and Variance of* Max *of Two Variables

The decision rule to represent the starting time of an activity, which will begin after the completion of two parallel activities is defined as:

$$\tilde{G}_2(\tilde{\mathbf{z}}) \leq max\{d_1^0, d_2^0\} + max\{\tilde{z}_1, \tilde{z}_2\}. \tag{4.6}$$

Note that we tighten the bound in Eqn 5.20 by replacing $\tilde{z}_1^+ + \tilde{z}_2^+$ with $max\{\tilde{z}_1, \tilde{z}_2\}$.

We now derive the expressions for expected value and variance of the adjustable variable, i.e., the RHS term of Eqn 4.6. Firstly, we focus on the expected value:

$$E[max\{d_1^0, d_2^0\} + max\{\tilde{z}_1, \tilde{z}_2\}] = max\{d_1^0, d_2^0\} + E[max\{\tilde{z}_1, \tilde{z}_2\}]. \tag{4.7}$$

In the general case, it is difficult to derive an exact expression for $E[max\{\tilde{z}_1, \tilde{z}_2\}]$ and hence, we provide an upper bound.

In the following Propositions 1 and 2, we compute expected value and variance for the more general case of $E(\tilde{z}) \geq 0$ (note that we assume $E(\tilde{z}) = 0$ for all primitive random variables). We calculate for the more general case because it will be required in the computation of expected value and variance for more than two random variables (next subsection).

Proposition 1. *The expected value for the maximum of two general distributions, $\tilde{z}_1$ and $\tilde{z}_2$ with nonnegative means is less than $\frac{1}{2}(E[\tilde{z}_1] + E[\tilde{z}_2]) + \frac{1}{2}\sqrt{Var[\tilde{z}_1] + Var[\tilde{z}_2] + (E[\tilde{z}_1])^2 + (E[\tilde{z}_2])^2}$.*

Proof. We begin by considering the following two equalities:

$$max\{\tilde{z}_1, \tilde{z}_2\} + min\{\tilde{z}_1, \tilde{z}_2\} = \tilde{z}_1 + \tilde{z}_2$$
$$max\{\tilde{z}_1, \tilde{z}_2\} - min\{\tilde{z}_1, \tilde{z}_2\} = |\tilde{z}_1 - \tilde{z}_2|.$$

We now sum the above two equalities.

$$max\{\tilde{z}_1, \tilde{z}_2\} = \tfrac{1}{2}(\tilde{z}_1 + \tilde{z}_2 + |\tilde{z}_1 - \tilde{z}_2|). \tag{4.8}$$

Thus, we can now compute the expected value of the maximum using the following equation:

$$E[max\{\tilde{z}_1, \tilde{z}_2\}] = \tfrac{1}{2}(E[\tilde{z}_1] + E[\tilde{z}_2] + E|\tilde{z}_1 - \tilde{z}_2|). \tag{4.9}$$

In addition, by using the definition of variance, we obtain:

$$Var|\tilde{z}_1 - \tilde{z}_2| = E(\tilde{z}_1 - \tilde{z}_2)^2 - (E|\tilde{z}_1 - \tilde{z}_2|)^2 \geq 0.$$

Therefore,

$$\begin{aligned} E|\tilde{z}_1 - \tilde{z}_2| &\leq \sqrt{E(\tilde{z}_1 - \tilde{z}_2)^2} \\ &= \sqrt{E(\tilde{z}_1^2) + E(\tilde{z}_2^2) - 2E(\tilde{z}_1)E(\tilde{z}_2)} \\ &\leq \sqrt{E(\tilde{z}_1^2) + E(\tilde{z}_2^2)} \\ &= \sqrt{Var[\tilde{z}_1] + Var[\tilde{z}_2] + E(\tilde{z}_1)^2 + E(\tilde{z}_2)^2}. \end{aligned} \tag{4.10}$$

Substituting the final expression of Eqn 4.10 into Eqn 4.9 yields the bound

$$E[max\{\tilde{z}_1, \tilde{z}_2\}] \leq \tfrac{1}{2}(E[\tilde{z}_1] + E[\tilde{z}_2]) + \tfrac{1}{2}\sqrt{Var[\tilde{z}_1] + Var[\tilde{z}_2] + (E[\tilde{z}_1])^2 + (E[\tilde{z}_2])^2}. \tag{4.11}$$

Hence the proof.

Note that in this work, we assume $E(\tilde{z}) = 0$, thus, a tighter bound can be obtained from Eqn 4.11:

$$E[max\{\tilde{z}_1, \tilde{z}_2\}] \leq \tfrac{1}{2}\sqrt{Var[\tilde{z}_1] + Var[\tilde{z}_2]}. \tag{4.12}$$

In the special case where $\{\tilde{z}_i\}$ $(i = 1, ...k)$ are normally and identically distributed, i.e. $\tilde{z}_i \sim N(0, \sigma)$, we know from the work of [11] that there is a closed form representation for the expected value of the maximum when $k = 2$:

$$E[max\{\tilde{z}_1, \tilde{z}_2\}] = \tfrac{\sigma}{\sqrt{\pi}}.$$

Now we focus on deriving expressions for variance of the maximum of two general distributions, i.e., $Var[max(\tilde{z}_1, \tilde{z}_2)]$.

Proposition 2. *The variance for the maximum of two general distributions, $\tilde{z}_1$ and $\tilde{z}_2$ with nonnegative means is less than $Var(\tilde{z}_1) + Var(\tilde{z}_2) + \frac{1}{2}(E(\tilde{z}_1))^2 + \frac{1}{2}(E(\tilde{z}_2))^2$.*

Proof. From Eqn 4.8, we have

$$\begin{aligned} Var[max(\tilde{z}_1, \tilde{z}_2)] &= \tfrac{1}{4} Var[\tilde{z}_1 + \tilde{z}_2 + |\tilde{z}_1 - \tilde{z}_2|] \\ &= \tfrac{1}{4}(Var[\tilde{z}_1 + \tilde{z}_2] + Var|\tilde{z}_1 - \tilde{z}_2| + 2COV(\tilde{z}_1 + \tilde{z}_2, |\tilde{z}_1 - \tilde{z}_2|)) \\ &\leq \tfrac{1}{4}(Var[\tilde{z}_1 + \tilde{z}_2] + Var|\tilde{z}_1 - \tilde{z}_2| + 2\sqrt{Var[\tilde{z}_1 + \tilde{z}_2] Var|\tilde{z}_1 - \tilde{z}_2|}) \\ &\leq \tfrac{1}{2}(Var[\tilde{z}_1 + \tilde{z}_2] + Var|\tilde{z}_1 - \tilde{z}_2|). \end{aligned} \tag{4.13}$$

Firstly, we consider the following two equations.

$$\begin{aligned} Var|\tilde{z}_1 - \tilde{z}_2| &= E(\tilde{z}_1 - \tilde{z}_2)^2 - (E|\tilde{z}_1 - \tilde{z}_2|)^2 \\ Var(\tilde{z}_1 - \tilde{z}_2) &= E(\tilde{z}_1 - \tilde{z}_2)^2 - (E(\tilde{z}_1 - \tilde{z}_2))^2 \end{aligned}$$

Subtracting the second from the first yields

$$Var|\tilde{z}_1 - \tilde{z}_2| = Var(\tilde{z}_1 - \tilde{z}_2) + (E(\tilde{z}_1 - \tilde{z}_2))^2 - (E|\tilde{z}_1 - \tilde{z}_2|)^2.$$

Now, we substitute this expression into the last term of Eqn 4.13 to obtain:

$$Var[max(\tilde{z}_1, \tilde{z}_2)] \leq Var(\tilde{z}_1) + Var(\tilde{z}_2) + \tfrac{1}{2}(E(\tilde{z}_1) - E(\tilde{z}_2))^2 - \tfrac{1}{2}(E|\tilde{z}_1 - \tilde{z}_2|)^2. \tag{4.14}$$

When no specific distribution about duration perturbation is known, we can obtain a bound for $Var[max(\tilde{z}_1, \tilde{z}_2)]$ as:

$$Var[max(\tilde{z}_1, \tilde{z}_2)] \leq Var(\tilde{z}_1) + Var(\tilde{z}_2) + \tfrac{1}{2}(E(\tilde{z}_1))^2 + \tfrac{1}{2}(E(\tilde{z}_2))^2. \tag{4.15}$$

Hence the proof.

Note that in this work, we assume $E(\tilde{z}) = 0$, thus, a tighter bound can be obtained from

Eqn 4.15:

$$Var[max(\tilde{z}_1, \tilde{z}_2)] \leq Var(\tilde{z}_1) + Var(\tilde{z}_2). \tag{4.16}$$

It is interesting to consider the special case when both random variables are normally distributed. We first state the following lemma[1].

Lemma 4.2.1. *If X is normally distributed* $X \sim N(0, \sigma)$*, then* $Y = |X|$ *is half-normally distributed, with*

$$E(Y) = \sigma\sqrt{\tfrac{2}{\pi}}. \tag{4.17}$$

Under normal distribution $\tilde{z}_i \sim N(0, \sigma_i)$, since $\tilde{z}_1 - \tilde{z}_2$ is also normally distributed, and $\tilde{z}_1 - \tilde{z}_2 \sim N(0, \sigma_1 + \sigma_2)$, we can conclude from Lemma 4.2.1 that $|\tilde{z}_1 - \tilde{z}_2|$ follows half-normal distribution with

$$E|\tilde{z}_1 - \tilde{z}_2| = (\sigma_1 + \sigma_2)\sqrt{\tfrac{2}{\pi}}. \tag{4.18}$$

Thus, if we substitute this expression into Eqn 4.14, we can express an upper bound on the variance value for the maximum duration perturbation of two activities, when $\tilde{z}_i \sim N(0, \sigma_i)$ as:

$$Var[max(\tilde{z}_1, \tilde{z}_2)] \leq (1 - \tfrac{1}{\pi})(\sigma_1^2 + \sigma_2^2) - \tfrac{2}{\pi}\sigma_1\sigma_2. \tag{4.19}$$

Expected Value and Variance of *Max* of Multiple Variables

Extending from two to k ($k > 2$) parallel activities, the completion time can be upper bounded by:

$$\tilde{G}_k(\tilde{\mathbf{z}}) \leq \max_{i=1,\ldots k}\{d_i^0\} + \max_{i=1,\ldots k}\{\tilde{z}_i\}. \tag{4.20}$$

In the following, we first compute the variance value of the above RHS term and then use a similar procedure to compute the expected value. The basic expression for variance of RHS is:

$$Var[\max_{i=1,\ldots k}\{d_i^0\} + \max_{i=1,\ldots k}\{\tilde{z}_i\}] = Var[\max_{i=1,\ldots k}\{\tilde{z}_i\}]. \tag{4.21}$$

To obtain the value of $Var[\max_{i=1,\ldots k}\{\tilde{z}_i\}]$ for general probability distributions, we take advantage of the analysis provided for the two-parallel-activity case above. The following steps

[1]This can be found in statistics texts, and found online at http://en.wikipedia.org/wiki/Half-normal_distribution.

outline the overall idea:

(a) Firstly, we group the activity set $\{a_1, ..., a_k\}$ into a couple set $\{C_1, ..., C_{\lceil \frac{k}{2} \rceil}\}$, where each element $C_j (j = 1, ... \lceil \frac{k}{2} \rceil)$ contains two different activities $C_j = \{a_{j1}, a_{j2}\}$ chosen from the activity set. Note that when k is an odd, the final element in the couple set contains just one activity.

(b) For each couple C_j, we apply the maximum operator on duration perturbations of involving activities. Denote $\tilde{c}_j = max\{\tilde{z}_{j1}, \tilde{z}_{j2}\}$, where $\tilde{z}_{j1}$ and $\tilde{z}_{j2}$ are duration perturbations of the two activities involved in C_j, then $Var(\tilde{c}_j)$ can be calculated based on the expression for the two-parallel-activity case.

(c) Then we have $\max\limits_{i=1,...k} \{\tilde{z}_i\} = \max\limits_{j=1,...\lceil \frac{k}{2} \rceil} \{\tilde{c}_j\}$. (Note again just one activity is contained in $C_{\lceil \frac{k}{2} \rceil}$ when k is odd). Then, we can build another couple set from $\{C_1, ..., C_{\lceil \frac{k}{2} \rceil}\}$, and the same method from steps (1) and (2) above is used to compute $Var[\max\limits_{j=1,...\lceil \frac{k}{2} \rceil} \{\tilde{c}_j\}]$ based on Eqn 4.15 and/or Eqn 4.16 and/or Eqn 4.19.

There are numerous ways (exponential in k) for generating the couple set $\{C_1, ..., C_{\lceil \frac{k}{2} \rceil}\}$ for k activities in parallel. Each of these couple sets can lead to different levels of tightness of derived robust makespan. To compute the grouping which provides the best robust fitness for random variables with generic distributions is an open problem. Instead, we focus on a heuristic that computes the best grouping under normal distribution $\tilde{z}_i \sim N(0, \sigma_i)$. It is obtained by solving the following optimization problem:

$$\max_{t} \sum_{j=1,...\lfloor \frac{k}{2} \rfloor} \sigma_{j1}\sigma_{j2} \tag{4.22}$$

where t denotes the grouping technique and is also the decision variable; $\{C_j\}$ is the couple set constructed from the activity set under grouping method t; σ_{j1} and σ_{j2} are the standard deviations of data perturbation for durations of activities contained in C_j. The intuition for employing this optimization problem is obtained from the Equation 4.19. It should be noted that computing a tighter bound on variance implies considering the highest possible value of the product of primitive variances. Hence, the reason for employing the optimization problem of Equation 4.22.

Proposition 3. *The solution t^* to the optimization problem of Eqn 4.22 is obtained by ordering*

the k activities in a non-increasing order of their variance values and then grouping all two nearest activities according to the order, i.e. $C_j = \{a_{j1}, a_{j2}\}$, where $j = 1, ...\lfloor \frac{k}{2} \rfloor$ and the standard deviations are in the following order:

$$\sigma_{11} \geq \sigma_{12} \geq \sigma_{21} \geq \sigma_{22} \geq, ...\sigma_{\lfloor \frac{k}{2} \rfloor 1} \geq \sigma_{\lfloor \frac{k}{2} \rfloor 2}. \tag{4.23}$$

Proof. Suppose we have another grouping method t', in which all elements in the couple set are the same as under t^* except two couples [2] where the ordering is different, i.e., $C_m = \{a_{m1}, a_{n2}\}$ and $C_n = \{a_{m2}, a_{n1}\}$ $(m \neq n)$, where $C_m = \{a_{m1}, a_{m2}\}$ and $C_n = \{a_{n1}, a_{n2}\}$ under t^*. Without loss of generality, assume $m > n$ and from Eqn 4.23, we have

$$\sigma_{m1} \geq \sigma_{m2} \geq \sigma_{n1} \geq \sigma_{n2}. \tag{4.24}$$

Since t' is supposed to provide a solution which is no less (defined in Eqn 4.22) than t^*, i.e.

$$\begin{aligned} &\sigma_{11}\sigma_{12} + ... + \sigma_{m1}\sigma_{n2} + ... + \sigma_{n1}\sigma_{m2} + ... + \sigma_{\lfloor \frac{k}{2} \rfloor 1}\sigma_{\lfloor \frac{k}{2} \rfloor 2} \\ &\geq \sigma_{11}\sigma_{12} + ... + \sigma_{m1}\sigma_{m2} + ... + \sigma_{n1}\sigma_{n2} + ... + \sigma_{\lfloor \frac{k}{2} \rfloor 1}\sigma_{\lfloor \frac{k}{2} \rfloor 2}. \end{aligned}$$

Therefore, we have

$$\sigma_{m1}\sigma_{n2} + \sigma_{n1}\sigma_{m2} \geq \sigma_{m1}\sigma_{m2} + \sigma_{n1}\sigma_{n2},$$

which is equivalent to: $(\sigma_{m1} - \sigma_{n1})(\sigma_{n2} - \sigma_{m2}) \geq 0$.
This contradicts Eqn 4.24 (except the case where all standard deviations are equal, in which case mixing the order does not affect anything). Thus, there exists no such t' which is different from t^* by at least two couples and has better objective value. The general case that t' has multiple (more than two) couples different from t^* can be easily derived from to this case (and is omitted due to space constraints).

Hence the proof.

As for analyzing the expected value $E[\max_{i=1,...k}\{\tilde{z}_i\}]$, we apply the same procedure employed to calculate the variance, i.e., based on the group solution returned by the above optimization

[2] It should be noted that if there is an ordering change in only one couple, then the method still produces the same solution because within a couple the variance computation does not consider the order.

problem, we first calculate the expected value for each couple and then, get the final bound following Eqn 4.11 and/or Eqn 4.12 and/or Eqn 4.13.

At present, we are unable to show the effectivness of our grouping heuristic (Equation 4.22) analytically in the most general case. However, we show the intuition behind the grouping heuristic by providing an analytical comparison[3] on an example where there are four activities (normally distributed durations) executed in parallel, i.e. $\tilde{z}_i \sim N(0, \sigma_i)$, and we assume $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \sigma_4$ (no loss of generality).

The representation of makespan under our grouping heuristic (denoted as M_{heu}) and random grouping (denoted as M_{ran}) are, respectively:

$$\begin{aligned} M_{heu} &= max\{d_1^0, d_2^0, d_3^0, d_4^0\} + max\{max\{\tilde{z}_1, \tilde{z}_2\}, max\{\tilde{z}_3, \tilde{z}_4\}\} \\ M_{ran} &= max\{d_1^0, d_2^0, d_3^0, d_4^0\} + max\{max\{\tilde{z}_1, \tilde{z}_4\}, max\{\tilde{z}_2, \tilde{z}_3\}\}. \end{aligned} \tag{4.25}$$

Let us first examine mean and variance values of M_{heu}. From Eqn 4.12, we have

$$\begin{aligned} E(max\{\tilde{z}_1, \tilde{z}_2\}) &\leq \tfrac{1}{2}\sqrt{\sigma_1^2 + \sigma_2^2} \\ E(max\{\tilde{z}_3, \tilde{z}_4\}) &\leq \tfrac{1}{2}\sqrt{\sigma_3^2 + \sigma_4^2}. \end{aligned} \tag{4.26}$$

From Eqn 4.19, we have

$$\begin{aligned} Var[max(\tilde{z}_1, \tilde{z}_2)] &\leq (1 - \tfrac{1}{\pi})(\sigma_1^2 + \sigma_2^2) - \tfrac{2}{\pi}\sigma_1\sigma_2 \\ Var[max(\tilde{z}_3, \tilde{z}_4)] &\leq (1 - \tfrac{1}{\pi})(\sigma_3^2 + \sigma_4^2) - \tfrac{2}{\pi}\sigma_3\sigma_4. \end{aligned} \tag{4.27}$$

From Eqn 4.11, Eqn 4.15, Eqn 4.26 and Eqn 4.27, we can obtain bounds of mean and variance values of of M_{heu} are [4]:

$$\begin{aligned} &E(M_{heu}) \leq \\ &const + \tfrac{1}{4}(\sqrt{\sigma_1^2 + \sigma_2^2} + \sqrt{\sigma_3^2 + \sigma_4^2}) + \tfrac{1}{2}\sqrt{(\tfrac{5}{4} - \tfrac{1}{\pi})\textstyle\sum_{i=1}^4 \sigma_i^2 - \tfrac{2}{\pi}(\sigma_1\sigma_2 + \sigma_3\sigma_4)} \\ &Var(M_{heu}) \leq (\tfrac{9}{8} - \tfrac{1}{\pi})\textstyle\sum_{i=1}^4 \sigma_i^2 - \tfrac{2}{\pi}(\sigma_1\sigma_2 + \sigma_3\sigma_4). \end{aligned} \tag{4.28}$$

[3] The calculation will use the robust fitness function provided in Definition 5.21 introduced in Section 3.5.
[4] Note that $const$ in Eqn 4.28 and Eqn 4.29 is $max\{d_1^0, d_2^0, d_3^0, d_4^0\}$.

Similarly, mean and variance values of of M_{ran} can also be calculated,

$$\begin{aligned} &E(M_{ran}) \leq \\ &const + \tfrac{1}{4}(\sqrt{\sigma_1^2+\sigma_4^2}+\sqrt{\sigma_2^2+\sigma_3^2}) + \tfrac{1}{2}\sqrt{(\tfrac{5}{4}-\tfrac{1}{\pi})\textstyle\sum_{i=1}^{4}\sigma_i^2 - \tfrac{2}{\pi}(\sigma_1\sigma_4+\sigma_2\sigma_3)} \\ &Var(M_{ran}) \leq (\tfrac{9}{8}-\tfrac{1}{\pi})\textstyle\sum_{i=1}^{4}\sigma_i^2 - \tfrac{2}{\pi}(\sigma_1\sigma_4+\sigma_2\sigma_3). \end{aligned} \tag{4.29}$$

From Eqn 5.21, bounds of fitness of M_{heu} (denoted by Fit_{heu}) and M_{ran} (denoted by Fit_{ran}) can then be respectively represented as a function of RHS of Eqn 4.28 and Eqn 4.29. We then examine the difference value between the two bounds, $Fit_{heu} - Fit_{ran}$. Let us first compare the first term of RHS of mean values in Eqn 4.28 and Eqn 4.29, since

$$\begin{aligned} &(\sqrt{\sigma_1^2+\sigma_2^2}+\sqrt{\sigma_3^2+\sigma_4^2})^2 - (\sqrt{\sigma_1^2+\sigma_4^2}+\sqrt{\sigma_2^2+\sigma_3^2})^2 \\ &= 2\sqrt{\sigma_1^2\sigma_3^2+\sigma_2^2\sigma_4^2+\sigma_1^2\sigma_4^2+\sigma_2^2\sigma_3^2} - 2\sqrt{\sigma_1^2\sigma_3^2+\sigma_2^2\sigma_4^2+\sigma_1^2\sigma_2^2+\sigma_3^2\sigma_4^2} \end{aligned} \tag{4.30}$$

and from Proposition 3, we have

$$\sigma_1\sigma_4+\sigma_2\sigma_3 \leq \sigma_1\sigma_2+\sigma_3\sigma_4. \tag{4.31}$$

Thus,

$$\sigma_1^2\sigma_4^2+\sigma_2^2\sigma_3^2-(\sigma_1^2\sigma_2^2+\sigma_3^2\sigma_4^2) = (\sigma_1\sigma_4+\sigma_2\sigma_3)^2-(\sigma_1\sigma_2+\sigma_3\sigma_4)^2 \leq 0. \tag{4.32}$$

From Eqn 4.31, Eqn 4.32, Eqn 4.28 and Eqn 4.29, we have that the bounds of mean and variance values of M_{heu} are lower than M_{ran}. Given the robust fitness function in Eqn 5.21, we conclude that

$$Fit_{heu} - Fit_{ran} \leq 0 \tag{4.33}$$

which is independent of ε and σ. In other words, our grouping heuristic can provide tighter fitness bound than random grouping.

4.3 Robust Local Search Algorithm

In [38], a robust local search method is proposed by integrating decision rule approximation introduced by SLA in Section 3.4.2 with the robust fitness function introduced in Section 3.5 for the problems represented by RCPSP/max with durational uncertainty. In this section, we give a detailed introduction of the robust local search algorithm and by replacing SLA with GNLA, we can obtain a local search model guided by a new decision rule. Steps 1, 2, 5 and 6 are standard steps in a local search algorithm. Steps 3 and 4 represent our departure from standard local search to deal with uncertainty.

1. **Generate initial solution**

   This is usually obtained using a simple greedy heuristic.

2. **Generate neighborhood of solutions**

   Generate a pool of neighbor solutions from the current solution.

3. **Employ one of the decision rule approximations (SLA and GNLA) for all adjustable variables and check feasibility**

   For each candidate solution $\mathbf{x}$ in the solution pool, derive the coefficients $C_k(\mathbf{x})$ for each adjustable variable. Subsequently, for each solution check constraint violation and reject those that are not feasible.

4. **Evaluate robust fitness function** f

   For each feasible solution $\mathbf{x}$, evaluate f to obtain the robust objective values. The solution with the lowest robust objective value is the current best robust solution.

5. **Apply penalty (optional)**

   Some advanced local search strategies may require a penalty to be applied to prevent it from being caught at a local minima. In the case of tabu-search for example, a tabu-list is updated when a tabu move is applied. In the case of iterated local search, a perturbation move will be applied to the current local minima.

6. **Termination criteria**

   If the termination criteria is met, return the solution with the lowest robust fitness function value else repeat the optimization cycle by determining the next move.

Algorithm 1 provides the robust local search algorithm guided by decision rule using SLA. By substituting $S^*_{now}, S^*_{min}, S^*$ with $G^*_{now}, G^*_{min}, G^*$, we obtain the local search algorithm using GNLA. Given the RCPSP/max with durational uncertainty and the level of risk ($0 < \epsilon \leq 1$), the algorithm returns the POS with the (locally) minimal robust makespan, S^* (or G^* by GNLA). In essence, we perform robust local search on the neighborhood set of activity lists. An activity list (al) is defined as a precedence-constraint feasible sequence that is used by heuristics to generate earliest start time schedules in solving the standard RCPSP problem [30].

Different activity lists are explored by local moves. In our context, we only consider the activity list as the sequence of activities which satisfy the non-negative minimal time lag constraint. Due to the existence of maximal time lag constraint in RCPSP/max, scheduling activities to their earliest possible start time based on the order position in the activity list may restrict the schedule so much that it may not even return in a feasible schedule. Thus, when we schedule each activity sequentially based on order position in the activity list, we will assign its starting time by randomly picking a time from its domain of feasible start times.

According to our experiments, this new randomized approach returns more feasible solutions than the earliest start time one. After finding a feasible schedule, a POS will be generated by applying the chaining procedure proposed by [49]. Then, the S^* (or G^* by GNLA) value will be computed according to the POS. Intuitively, using the randomized approach may return a schedule with a large baseline scheduled completion time. However, we can apply the shortest path algorithm on the resulting POS to generate the earliest start time schedule for a smaller makespan.

As mentioned above, it may be difficult to find a feasible schedule that satisfies minimal and maximal time lag constraints using the activity list. In fact, we believe that in the set of all activity lists, many may not yield a feasible schedule. We overcome this problem as follows. We define the set of activity lists which result in feasible (or infeasible) schedules as F (or I). We seek to design a local search algorithm with the following characteristics: a) Starting from an activity list in I, the local search should move to an activity list in F within a short time. b) Starting from an activity list in F, the local search should move to the activity list with the minimal S^*(or G^* by GNLA)value. c) We also diversify the exploration of activity lists in F by allowing the local search to move from an activity list in F to an activity list in I, since activity

Algorithm 1 Robust Local Search

[38]

Generate an activity list **al** randomly
Find a start time schedule, ss randomly according to **al**
if al $\in F$ **then**
    $POS \leftarrow chaining(ss)$
    Compute S^*_{now} according POS
    Update S^*_{min} as S^*_{now}
else
    Record the first activity a which cannot be scheduled
end if
for $i \leftarrow 1$ to Max_Iteration **do**
    if al $\in I$ **then**
        Shift activity a ahead in **al** randomly as **al**'
    else
        Select two activities b and c in **al** randomly
        Swap b and c in **al** as **al**'
    end if
    Find randomized start time schedule ss' according to **al**'
    if al$' \in F$ **then**
        $POS' \leftarrow chaining(ss')$
        Compute S^* according to POS'
        if al $\in I$ or $S^* \leq S^*_{now}$ **then**
            $S^*_{now} \leftarrow S^*$
            al $\leftarrow$ **al**$'$
            if $S^* \leq S^*_{min}$ **then**
                $S^*_{min} \leftarrow S^*$
            end if
        end if
    else if al $\in I$ **then**
        al $\leftarrow$ **al**$'$
    else
        $p \leftarrow rand(0, 1)$
        if $p < 0.01$ **then**
            al $\leftarrow$ **al**$'$
            Record the first activity a which cannot be scheduled
        end if
    end if
end for

lists in the F region may not be reachable from one another by simple local moves. This has the flavor of strategic oscillation proposed in meta-heuristics research.

The detailed robust local search procedure is given in Algorithm 1. The procedure starts by randomly generating an activity list **al**, which is a sequence of activities that satisfy the non-negative minimum time lag constraint (Line 1). In Line 2, a schedule *ss* is produced based on ordering of activities in the activity list **al**. We first perform domain reduction on the distance graph using the Floyd-Warshall algorithm, so that the feasible range of the start time for each activity based on the temporal constraints can be obtained. We then schedule each activity sequentially based on the order position in the activity list. For each activity, we first pick a start time randomly from the feasible domain and evaluate resource constraints for the duration of the activity (i.e. check if the current resource capacity exceeds the resource amount used by that activity). If yes, we set the start time to that activity, run the shortest path algorithm to reduce domains for the remaining activities, and update current resource capacity due to consumption of that activity. If the resource constraints are not satisfied, we will try to set the start time randomly again for a prescribed maximum numbers of retries. Once the start time of current activity is set, we proceed iteratively to the next activity according to the activity list. In Line 4, *chaining*() is employed to generate a POS from a baseline schedule (section 3.2.1). $Max\_Iteration$ refers to the maximum number of iterations in the robust local search. We apply two different types of local moves. To converge quickly to an activity list in F, the first local move is designed to schedule the activity that is causing a temporal or resource conflict to an earlier time. It will randomly shift ahead the first activity which cannot be scheduled in the current activity list (Line 12). When an activity list is in F, the second local move will randomly pick two activities and swap them in the current activity list, while satisfying the non-negative minimal time lag constraints (Line 14-15). The move will be accepted, if it results in a smaller or equal S^* value (Line 18-29). To explore different activity lists, we include a small probability to accept the move which leads to an infeasible schedule (Line 31-35). The probability to move from an activity list in F to one in I is set at 0.01. The minimal S^* value will be saved as S^*_{min}.

The worst-case computational complexity analysis is given as follows. For each iteration in local search, there are three major components: randomized schedule generation, POS construction and fitness calculation. In the process of randomized schedule generation, we per-

form domain reduction and resource checking at each iteration, and thus the complexity is $O(N \cdot (N^3 + H \cdot K \cdot w))$ where N is the number of activities, H is the maximum planning horizon, K is the number of types of resources, and w is the prescribed maximum number of retries for each activity on setting the randomized start time. The POS construction process works as follows: the set of activities are first sorted according to their start times in the generated deterministic schedule and the sorting part costs $O(N \cdot logN)$; then it proceeds to allocate each activity the total units needed for each type of resource. Let max_{cap} be the maximum capacity among all resources. The cost for computing POS is then $O(N \cdot logN + N \cdot K \cdot max_{cap})$. When determining the fitness value of generated POS, we examine edge by edge to check if it is connected in parallel or in serial with respect to its predecessors and it costs $O(N + e)$ where e is the number of edges in POS ($e < N^2$). Thus, the worst-case complexity of our proposed robust local search algorithm is $O(TN \cdot (N^3 + H \cdot K \cdot w + K \cdot max_{cap}))$ where T is the number of iterations in local search.

4.3.1 Schedule Infeasibility of a Given POS

It should be noted that the fitness function, f assumes that any schedule generated by the POS, $\mathbf{x}$ is always executable. However, due to durational uncertainty and the maximum time lags, the schedule is not always executable. A direct way to measure IPr(POS) the probability of infeasibility of the POS (i.e. probability that the POS can lead to an infeasible schedule) lies in the computation of the probability of infeasibility of each activity a_i $IPr(a_i)$, that there does not exist a feasible start time such that all temporal constraints with respect to a_i are satisfied. IPr(POS) can be calculated as the probability that at least one activity is infeasible. However, due to temporal dependencies between activities providing a theoretical expression for the overall probability of infeasibility is an open problem. Therefore, we propose a simulation approach, where we simulate POS execution over multiple trials to compute this probability eciently and approximately. As an illustration, we experimented with the benchmark J10 instances from the PSPLib [31] for RCPSP/max with additional durational uncertainty that follows a normal distribution with mean 0 and variance 1. We generated 1000 sample realizations for the POS obtained from SLA, and check for infeasibility with respect to the original temporal (including the maximum time lag) constraints. Examples of the probability of infeasibility

obtained by our simulation for PSP1, PSP4, and PSP13 are 0.18, 0.17 and 0.001. However, for the other problems PSP3, PSP5 etc. the probability of infeasibility was 0, because the maximal time lags were much larger than the variance of durational uncertainty.

4.4 Enhancing Robust Local Search

In this section, we describe two enhancements to improve the basic local search method described in Section 4.3. Firstly, we describe ordering generation, which is a pre-processing step used to identify precedence ordering between activities. This precedence ordering is then used to focus the local search over activity lists. Secondly, we describe a new chaining method to generate POS from a feasible schedule.

4.4.1 Ordering Generation

Ordering Generation is a pre-processing step that identifies precedence relationships between pairs of activities. The key idea is that for certain pairs of activities, it is always better (with respect to robust makespan) to have the same ordering among activities. Our goal is to identify these pairs of activities and employ this ordering to focus the local search over activity lists and in the chaining method used to compute POS from feasible schedule.

In deciding an ordering between a pair of activities, a and b, there are two key steps: (i) Sample set generation: Generate two sets of m activity lists. The first set consists of m activity lists where a occurs before b. The second set is generated by swapping activities a and b in every activity list in the first set; (ii) Order determination: In this step, we first compute POS and its robust makespan for all activity lists in the two sets. By comparing the robust makespan values of corresponding activity lists in the two sets, we determine an ordering between activities. We explain these steps in the following subsections.

For a problem with *n* activities, there are C_n^2 pairs of activities. If we are to decide the orders between all pairs, the ordering computation needs to be implemented for C_n^2 times, which is computationally expensive. Based on this observation, we first propose a *Pairs-Selection* heuristic to selectively choose a certain number of activities pairs whose ordering can have a

significant impact on the robust makespan.

The *Pairs-Selection* heuristic picks an activity pair: (a) If it is not precedence related in the original problem definition; and (b) If there exists at least one type of resource, where the total demand of both activities exceeds the resource capacity. The intuition behind picking such an activity pair is that those two activities cannot be executed in parallel and deciding an ordering relationship is imperative to eliminate the resource conflict. One main advantage of the heuristic is that the number of pairs of activities that need to be ordered is significantly reduced. Now, we describe the two steps of ordering generation below:

Sample Set Generation

We first randomly generate m activity lists as an initial sample set denoted by T. Each element in T is an activity list represented as al_i which is a sequence of all activities, where $i = 1, ...m$, i.e.

$$T = \{al_i | al_i = (a_1, a_2, ...a_n), \forall i \in \{1, ...m\}\}.$$

For each pair of activities (a_k, a_l) resulting from the *Pairs-Selection* heuristic, we define two sample sets represented as $T^{a_k \prec a_l}$ and $T^{a_l \prec a_k}$. $T^{a_k \prec a_l}$ has all the activity lists that are in T, except that if an activity list has a_l before a_k, then those activities are swapped.

$$T^{a_k \prec a_l} = \{al_i^{a_k \prec a_l} | i \in \{1, ...m\}\},$$

$$\text{where } al_i^{a_k \prec a_l} = \begin{cases} (a_1, a_2, ..., a_k, ...a_l, ...a_n) & if\ al_i = (a_1, a_2, ..., a_l, ...a_k, ...a_n) \\ al_i & if\ \ al_i = (a_1, a_2, ..., a_k, ...a_l, ...a_n) \end{cases}.$$

Similarly, $T^{a_l \prec a_k}$ can be constructed by incrementally selecting each activity list from the initial set T with $a_l \prec a_k$ and reverse the order if $a_k \prec a_l$, i.e.

$$T^{a_l \prec a_k} = \{al_i^{a_l \prec a_k} | i \in \{1, ...m\}\},$$

$$\text{where } al_i^{a_l \prec a_k} = \begin{cases} (a_1, a_2, ..., a_l, ...a_k, ...a_n) & if\ al_i = (a_1, a_2, ..., a_k, ...a_l, ...a_n) \\ al_i & if\ \ al_i = (a_1, a_2, ..., a_l, ...a_k, ...a_n) \end{cases}.$$

Thus, each activity list in the sample set $T^{a_k \prec a_l}$ share the same positions of all activities except a_k and a_l with the corresponding activity list in set $T^{a_l \prec a_k}$, where a_l precedes a_k.

Order Determination

We then determine the activity order of each selected pair of activities based on the sample sets obtained from last phase. For a pair (a_k, a_l), we construct a new instance by posting a precedence constraint $a_k \prec a_l$ or $a_l \prec a_k$ to the original instance, and based on the new instance, we determine the fitness which are denoted as $f_i^{a_k \prec a_l}$ and $f_i^{a_l \prec a_k}$ for $al_i^{a_k \prec a_l}$ and $al_i^{a_l \prec a_k}$, respectively.

Note that $al_i^{a_k \prec a_l}$ and $al_i^{a_l \prec a_k}$ share the same elements and the same positions except the order of a_k and a_l. Thus, the order of a_k and a_l can be considered as a reason why the fitness of $al_i^{a_k \prec a_l}$ and $al_i^{a_l \prec a_k}$ differs. To decide the order of a_k and a_l, we define an *index variable* denoted as $iv_{a_k \prec a_l}$ that measures the percentage of samples where the one with the order a_k proceeds a_l wins, i.e.

$$iv_{a_k \prec a_l} = -\frac{\sum_i min(\frac{f_i^{a_k \prec a_l} - f_i^{a_l \prec a_k}}{|f_i^{a_k \prec a_l} - f_i^{a_l \prec a_k}|}, 0)}{m}.$$

We then define an *Index Parameter* for activities a_k and a_l denoted as $IP_{a_k \prec a_l}$ as a benchmark for the index variable $iv_{a_k \prec a_l}$ in determining the order of a_k and a_l. The parameter $IP_{a_k \prec a_l}$ can be prescribed by users and different values (usually larger than 50%) represent different levels of confidence that the order of a_k and a_l matters in causing fitness variance, and thus also represents different controllability of $iv_{a_k \prec a_l}$.

If the value of index variable $iv_{a_k \prec a_l}$ is larger than the value of $IP_{a_k \prec a_l}$, we set the order $a_k \rightarrow a_l$ since there indicates a higher probability that $a \rightarrow b$ can provide better robustness than $b \rightarrow a$; If $iv_{a_k \prec a_l}$ is less than $1 - IP_{a_k \prec a_l}$, we set $a_l \rightarrow a_k$; And in other cases, no order between a_k and a_l is settled.

4.4.2 Improved Chaining based on Robustness Feedback

As noted in the "Preliminaries" section, for each activity a, there may exist multiple choices of resource chains to which it can be assigned. In addition, different chaining heuristics will lead to POSes that can have different robust makespan values. In this section, we propose a new chaining heuristic that dispatches activities to resource chains by predicting the improvement in robust makespan of the generated POS.

Algorithm 2 Robustness-Feedback Resource Chaining (Activity a, Schedule S, Order G)

```
C ← Find set of available chains, C for activity a based on S
P ← Collect chains from C with last activity of chain preceding a in problem
O ← Collect chains from C with last activity of chain ordered before a in G
if P ≠ φ then
    k ← Get first available chain in P
else if O ≠ φ then
    k ← Get first available chain in O
else
    k ← Get first available chain in C
end if
Post constraint between last activity of chain k (denoted as last(k)) and activity a
if a requires more than one resource unit then
    C1 ← chains in C which have last activity as last(k)
    C2 ← C \ C1
    for all resource units required by a do
        choose the first available chain belonging to C1
        if chain above is not feasible then
            choose the first available chain belonging to C2
        end if
    end for
end if
```

In the latest chaining method which aims to increase flexibility as described in Section 3.2.1, the chains are first randomly picked from a superior subset (i.e., chains where the last activity is already ordered, or chains sharing the same last element). Since our objective is makespan-related and time becomes a concern, we build on the work of [47] and pick the *first* available chain wherever available. The updated chaining method is called *Robustness-Feedback based Resource Chaining*.

Example 5. *Figure 4.1 provides the POS provided by this chaining heuristic when used on Example 1. As can be seen, compared to the POS in 3.4, the key difference is the allocation of activity 5 and 6. With our new heuristic, it can be seen that there is more parallelism and hence reduced robust makespan with high probability.*

When employing the Ordering Generation algorithm in conjunction with the chaining heuristic, we also consider the information about ordered pairs when allocating resource units to an activity. The motivation is that once activity *a* and activity *b* (for example, $a \rightarrow b$) is ordered, there is a high probability that this precedence relationship can result in a better solu-

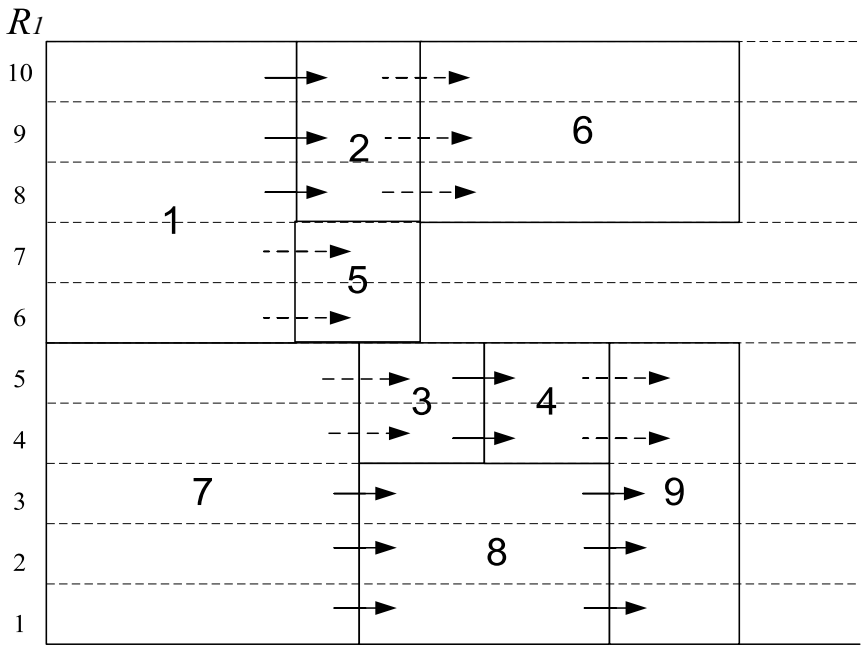


Figure 4.1: POS by Robustness Feedback Chaining

tion. Algorithm 2 provides the pseudo code for the Robustness-Feedback Resource Chaining heuristic with Ordering.

4.5 Experimental Evaluation

In this section, we first evaluate the scalability and quality of the execution strategies provided by robust local search and the various enhancements introduced in this work. Secondly, to establish a benchmark on the performance, we compare against the best known technique for solving JSP problems with durational uncertainty. It should be noted that the robust local search method is developed to solve RCPSP/max problems with durational uncertainty and hence does not exploit the structure present in JSP problems. Furthermore, as described earlier, the optimization metrics of both approaches are different.

4.5.1 Experimental Setup

We have two sets of problems that we consider and those are described in the subsections below. Additionally, we also indicate the algorithms that are compared on each of the data sets in this

section.

RCPSP/max with Durational Uncertainty

The problems considered for RCPSP/max with durational uncertainty were obtained by extending the three benchmark sets available for RCPSP/max problems, J10, J20 and J30 as specified in the PSPLib [31]. Each set contains 270 problem instances with duration for each activity ranging between 1 and 10. The maximum number of activities for J10, J20 and J30 are 10, 20 and 30, respectively. For each activity a_i, we set the expected value d_i^0 of the stochastic duration as the corresponding deterministic duration given by the benchmarks, and assume that duration uncertainty is normally distributed, i.e. $\tilde{z}_i \sim N(0, \sigma)$. Henceforth, we refer to J10, J20 and J30 as these RCPSP/max problems with durational uncertainty. We run the algorithms on problems with four different duration variabilities $\sigma = \{0.1, 0.5, 1, 2\}$ and four increasing levels of risk $\varepsilon = \{0.01, 0.05, 0.1, 0.2\}$.

On RCPSP/max problems with durational uncertainty, we compare the robust local search that is guided using the two decision rule approximations SLA and GNLA. Furthermore, we also compare the different enhancements to robust local search on RCPSP/max problems with durational uncertainty. We compare five different variants of robust local search for each decision rule approximation: (a) (GNLA) refers to basic robust local search guided by GNLA decision rule approximation; (b) (GNLA+RC) is the robust local search with the new Robustness-feedback Chaining heuristic guided by GNLA; (c) (GNLA+) refers to the basic robust local search with additional local search iterations, where the number of local search iterations is determined based on the problem set (as described later); (d) (GNLA+OG) is the Order Generation heuristic on top of GNLA guided robust local search; and finally (e) (GNLA+OG+RC) has both Order Generation and Robustness-feedback Chaining heuristics on GNLA guided robust local search.

The number of local search iterations for robust local search was set to 1000. To reduce the stochasticity effects of robust local search, we average over 10 random executions for each problem instance. Our code was implemented in C++ and executed on a Core(TM)2 Duo CPU 2.33GHz processor under FedoraCore 11 (Kernel Linux 2.6.29.4-167.fc11.i586).

JSP with Durational Uncertainty

For JSPs, (GNLA) is compared against the probabilistic makespan results provided by [5]. For the benchmark problems, we consider the instances generated using an existing generator in the work of [60] with durations drawn uniformly from the interval [1,99]. Specifically, we focus on three sets of probabilistic JSPs of size $\{4\times4,6\times6,10\times10\}$ (where a 4×4 problems consists of 4 jobs consisting of 4 activities each) and for each set, three uncertainty levels $\{0.1,0.5,1\}$ are considered.

4.5.2 Comparison between SLA and GNLA

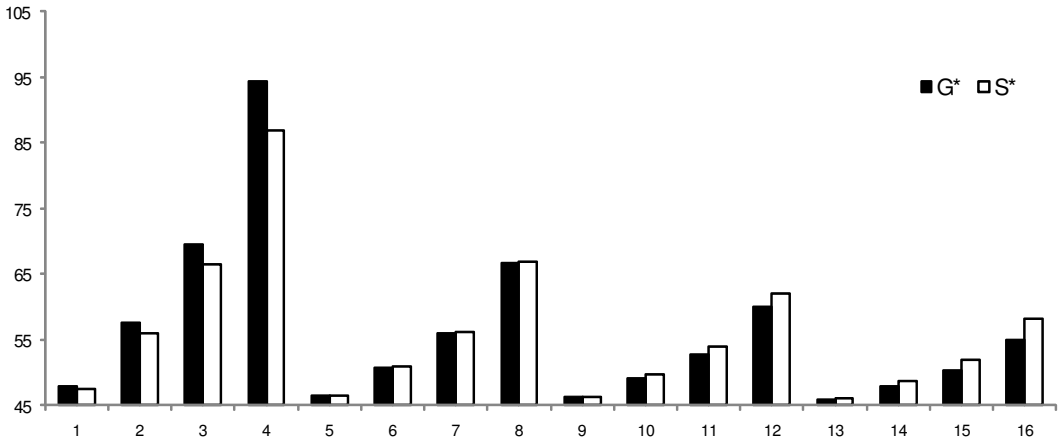
We first compare the average robust makespan of 270 problem instances obtained by robust local search that is guided by our decision rule approximations proposed in Section 3.4.2 and Section 4.2.1. We refer to the robust makespan computed using SLA as S^* and using GNLA as G^*. Figure 4.2 provides these results for all three sets of RCPSP/max problems with durational uncertainty. In these results, we show how the robust makespan is affected by the level of risk ε and the standard deviation σ of duration uncertainty. X-axis represents different combinations of risk and standard deviation of durational uncertainty, as shown in the table of Figure 4.2. All runs on every instance takes a couple of seconds and hence we do not report CPU times here. The key observations and conclusions of interest from Figure 4.2 are as follows:

- Irrespective of the σ, as the level of risk ε increases, the robust makespan decreases with both SLA and GNLA. Clearly, the lower risk that the planner is willing to take, the higher is the robust value of the generated execution strategy. Our method is capable of quantifying this trade off, which can help the planner to decide on the desired strategies.

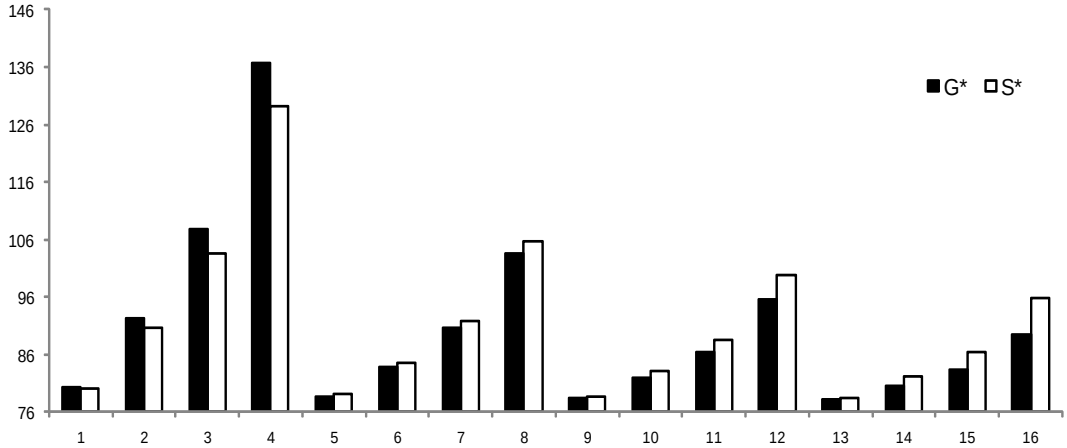
- Irrespective of ε, as the degree of duration variability σ increases, the robust makespan increases with both SLA and GNLA, and the value becomes more sensitive to σ when the level of risk is constrained to a small value (e.g. $\varepsilon = 0.01$).

- For lower values of ε, more specifically for 0.01, S^* provides lower values of robust makespan than G^*. On the other hand, for higher values of $\varepsilon \in \{0.05, 0.1, 0.2\}$, G^* provides superior performance to S^*. We do not yet understand the reason for drop in

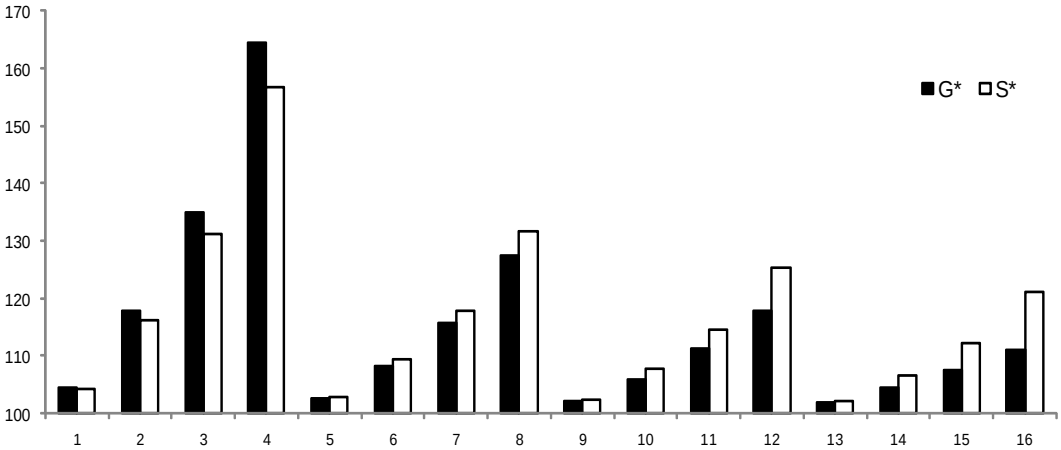
Horizontal Scale	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Sigma x Epsilon	0.1 x 0.01	0.5 x 0.01	1 x 0.01	2 x 0.01	0.1 x 0.05	0.5 x 0.05	1 x 0.05	2 x 0.05	0.1 x 0.1	0.5 x 0.1	1 x 0.1	2 x 0.1	0.1 x 0.2	0.5 x 0.2	1 x 0.2	2 x 0.2



(a) Results of J10



(b) Results of J20

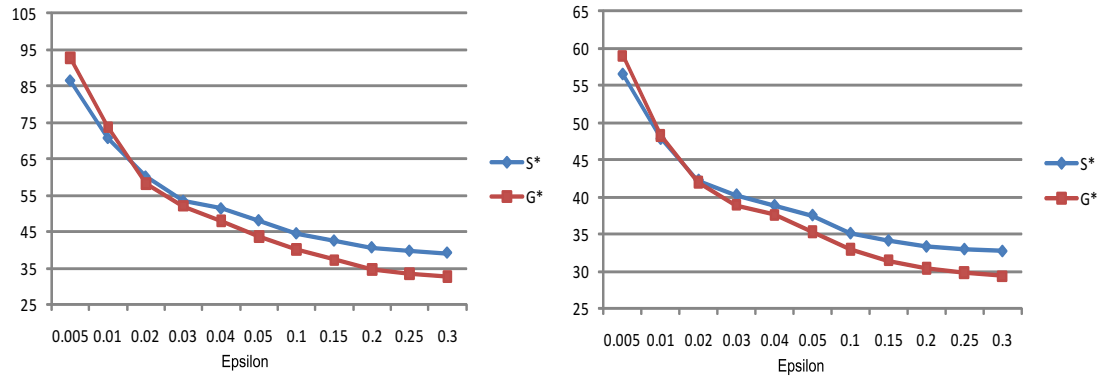


(c) Results of J30

Figure 4.2: Comparison of Robustness between SLA and GNLA.

performance for $\varepsilon = 0.01$, but this is observed consistently across all the RCPSP/max benchmark problems.

For each problem instance, we also observe some monotonicity between the absolute difference of robust makespan S^* and G^* and risk values. When the level of risk ε takes a value around 0.02, S^* (SLA) has a slightly lower value than G^* (GNLA). However, when risk becomes more than 0.02, the superiority of GNLA increases with higher values of risk. Figure 4.3 illustrates this on a randomly picked J10 instance with $\sigma = 1$ and $\sigma = 2$. The same pattern is observed across all problem instances of J10, J20 and J30.

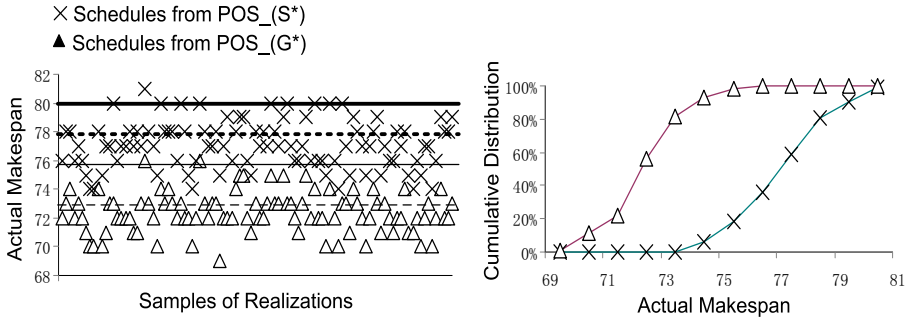


(a) Results from a randomly selected J10 example with σ=1

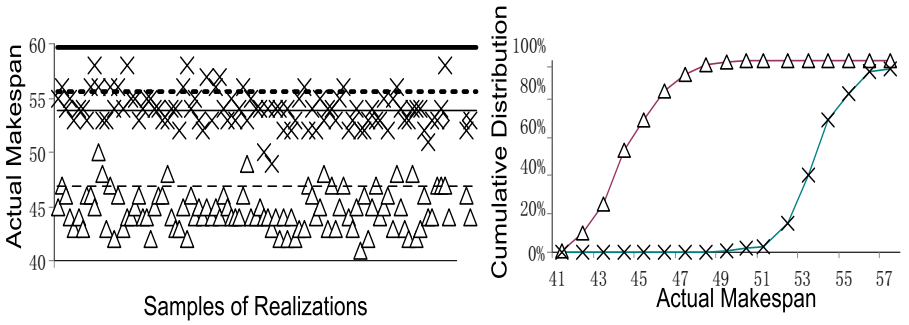
(b) Results from a randomly selected J10 example with σ=2

Figure 4.3: Comparison of Robust Makespan.

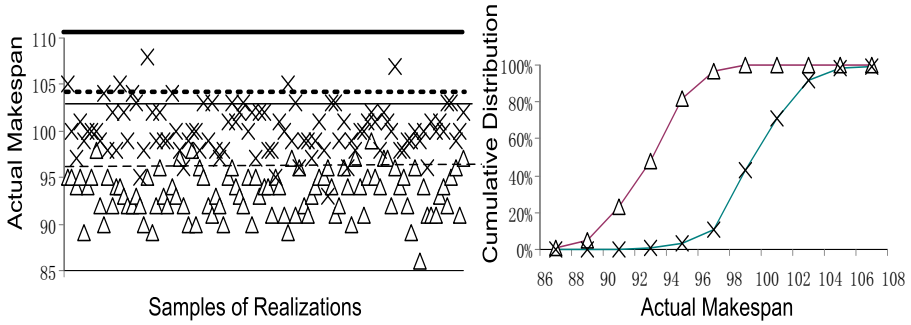
Next, in Figure 4.4, we compare the quality of the execution strategies obtained by using SLA and GNLA. More precisely, we compare the distributions of the actual makespans of schedules computed using these decision rule approximations. For this purpose, we generate a set of 100 samples of realizations of durational uncertainty and test with all 270 instances of each benchmark set with different levels of risk $\varepsilon = 0.2$, $\varepsilon = 0.1$ and $\varepsilon = 0.05$ to obtain the respective POS, and then compute the actual makespans of schedules derived from the respective POS under the given realization samples. This difference between real makespans obtained from POSs generated by two different decision rule approximations was observed across the board in all examples of the three sets for all values of ε except 0.01. We randomly select three problem instances from each benchmark set and present the results in Figure 4.4. Figure 4.4 also compares the cumulative frequency distributions of the actual makespans. We observe that GNLA provided far better realized makespans than SLA - both in absolute terms,



(a) Results from randomly selected J10 example with $\varepsilon = 0.2$



(b) Results from a randomly selected J20 example with $\varepsilon = 0.1$



(c) Results from a randomly selected J30 example with $\varepsilon = 0.05$

Figure 4.4: Comparison of Actual Makespans and Gap between S^* and G^*.(Lines in left pictures from top down indicating: Computed S^*, Actual S^* by Simulation, Computed G^*, Actual G^* by Simulation.)

as well as distributionally. For J20, except in 2 cases, rest of the actual makespan values obtained by SLA were higher than the ones obtained by GNLA. Similar trends were observed for J10 and J30.

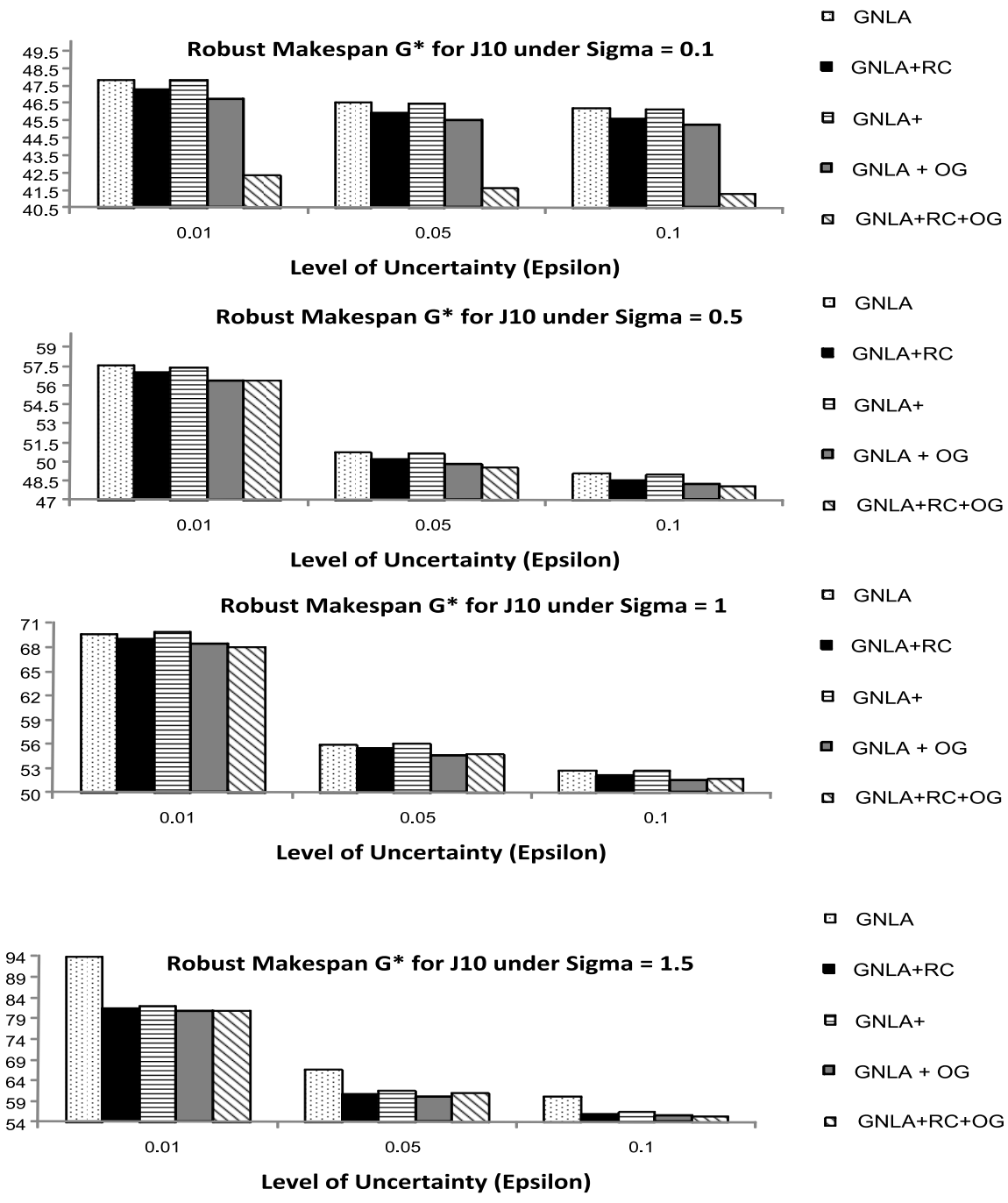
To illustrate the difference of quality in absolute of the two upper bounds, we provide four lines (computed S^*, actual S^*, computed G^* and actual G^*) indicating the upper bounds computed using the algorithms and in simulation over the 100 samples.

4.5.3 Comparisons between Robust Local Search Enhancements

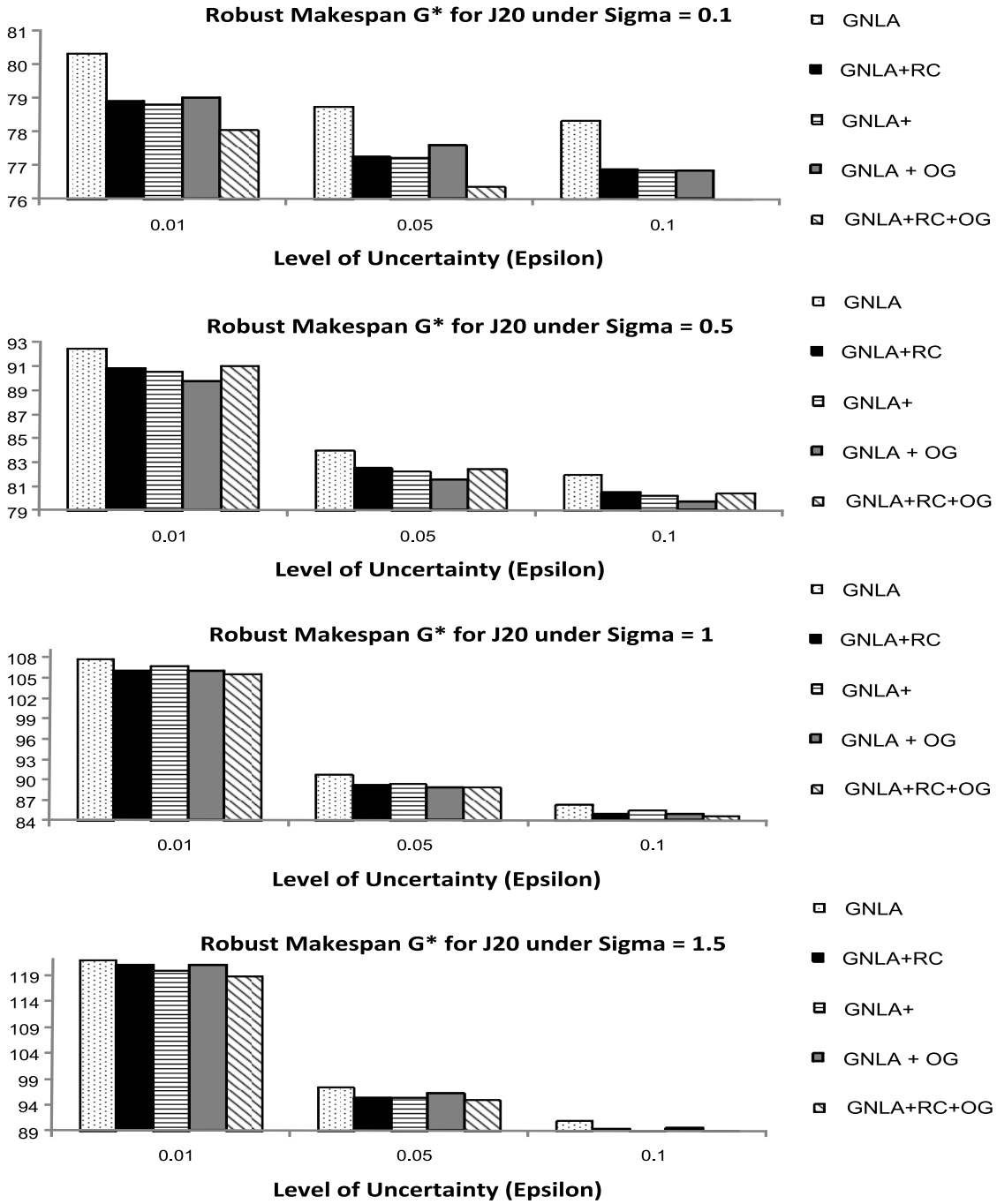
Since, we have already shown GNLA performs better than SLA, we will only show the performance of our enhancements over GNLA in this section. It should be noted that enhancements over SLA provided similar results and conclusions with GNLA based enhancements outperforming SLA based enhancements. Since "Ordering Generation" heuristic requires additional rounds of robust makespan computation, we also include a benchmark called (GNLA+) (which is GNLA plus extra iterations of local search) to make a fair comparison. To avoid the complexity of considering all pairs of activities, we only consider those pairs of activities where ordering would improve performance. We proposed the Pairs-Selection heuristic to select these pairs of activities. The number of extra iterations of local search for the (GNLA+) benchmark is "the number of activity pairs picked by the Pairs-Selection heuristic" times "the number of samples m used in the Ordering Generation process". The experimental results shows that the average number of activity pairs of all 270 instances selected under the Pairs-Selection heuristic for J10, J20 and J30 are 5, 14, and 28 respectively. In our work, we set $m = 100$. Thus, the extra iterations of the (GNLA+) benchmark for J10, J20 and J30 are 500, 1400 and 2800, respectively. The performance of all our enhancements is shown in Figure 4.5(a), Figure 4.5(b), Figure 4.5(c) for J10, J20 and J30 respectively. In all the charts, ε is represented on the X-axis and robust makespan on the Y-axis. So, lower values are better on the Y-axis.

Given below are some key observations and conclusions made from the results:

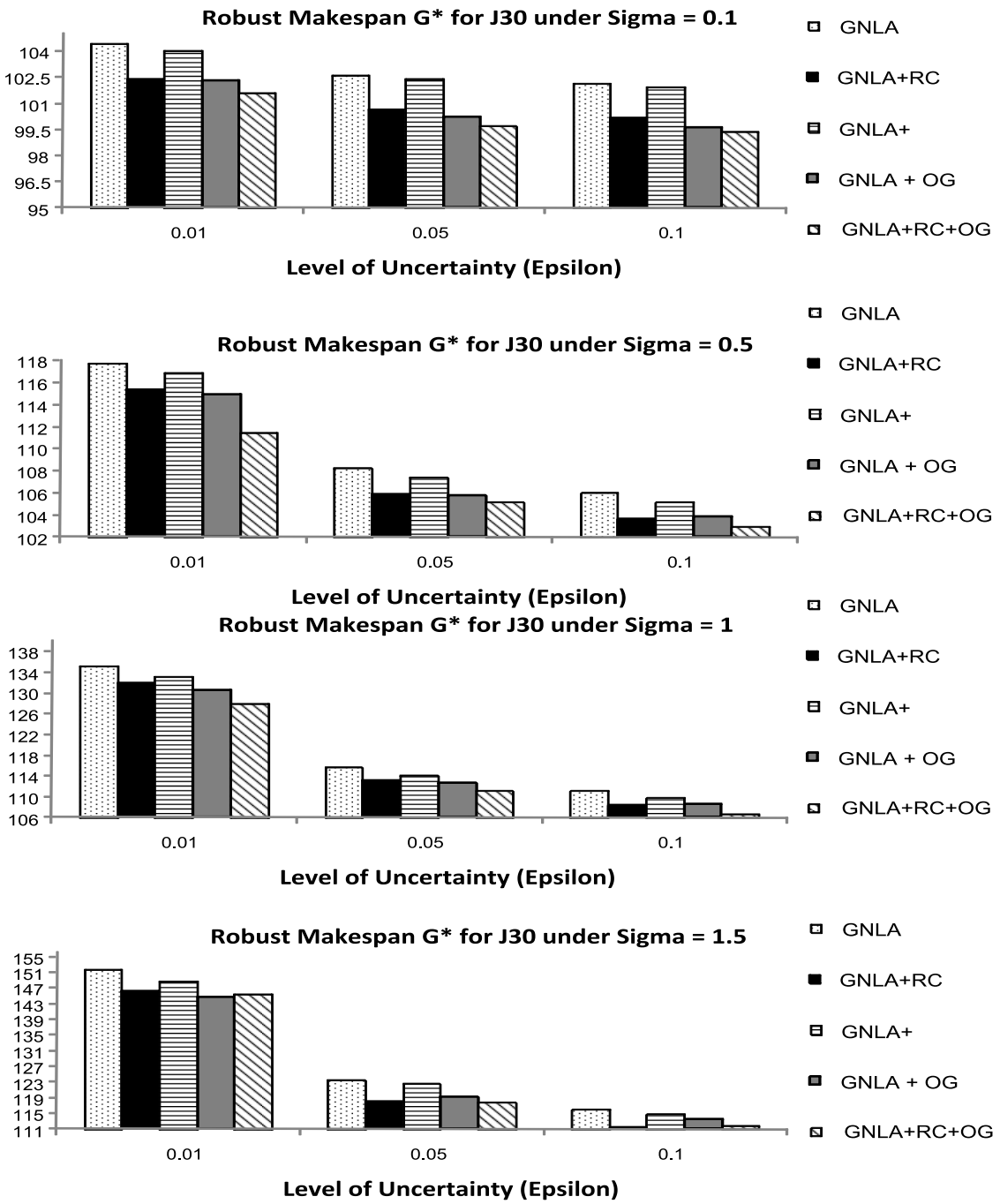
- Irrespective of the durational uncertainty, (GNLA+RC) and (GNLA+OG) provide better robust makespan values than both (GNLA) and (GNLA+) for J10 and J30. This indicates that the new Robustness Feedback Chaining heuristic and the Order Determination are



(a) Results of J10



(b) Results of J20



(c) Results of J30

Figure 4.5: Comparison of Robust Local Search Enhancements.

able to provide more robust partial ordered schedules for J10 and J30. This improvement seems to increase further with more number of activities, i.e. the difference is more obvious for instances in J30 than in J10. Furthermore, the difference is consistently observed across all the problems. However, the improvement is not consistent for J20 and there are cases where (GNLA+RC) and (GNLA+OG) did not out perform (GNLA) and (GNLA+). For instance in J20 problems, (GNLA+) provides better performance than (GNLA+RC) and (GNLA+OG) for $\varepsilon = 0.01$ and $\sigma = 1.5$.

- The extra iterations of local search in (GNLA+) do not improve the solution quality much for J10. However, it improves the solution quality for J20 and J30. This could be because the optimal solution is obtained within 1000 iterations for the smaller problems.

- In most cases, (GNLA+RC+OG) provides the lowest robust makespan among all the enhancements. Thus, the OG and RC enhancements in combination do not degrade the performance improvement obtained individually. In some cases, the difference is significant such as in J10 for $\sigma = 0.1$ and $\varepsilon = 0.01$. On the other hand, there are cases where (GNLA+RC+OG) does not provide the lowest robust makespan, such as in J20 for $\sigma = 0.5$ and $\varepsilon = 0.01$.

4.5.4 Comparisons with JSPs with Durational Uncertainty

In this section, we compare the performance of our GNLA approach (referred to as G^*) with the best known solver for Job Shop Scheduling Problems proposed by [5] (referred to as CB). For a fair comparison of the two approaches, we employ the Mean Normalized Makespan (MNPM) metric defined by Beck and Wilson:

$$MNPM(a, L) = \frac{1}{|L|} \sum_{l \in L} \frac{D(a,l)}{D_{lb}(l)} \tag{4.34}$$

where L is a set of problem instances, $D(a, l)$ is the probabilistic makespan (i.e., robust makespan in our work) for instance l by algorithm a generated by Monte Carlo simulation, $D_{lb}(l)$ is a lower bound on the probabilistic makespan.

We denote the best MNPM values aross different algorithms reported by [5] as CB. We

MNPM	Problem Size 4×4			Problem Size 6×6			Problem Size 10×10		
	UL=0.1	UL=0.5	UL=1	UL=0.1	UL=0.5	UL=1	UL=0.1	UL=0.5	UL=1
CB	1.023	1.046	1.128	1.021	1.073	1.168	1.024	1.101	1.215
G^*	1.066	1.123	1.282	1.095	1.190	1.273	1.210	1.225	1.263

Table 4.1: Comparison against CB solver(UL:Uncertainty Level).

compare them with the MNPM values in our work which are obtained by replacing $D(a, l)$ in Eqn 4.34 with an upper bound of robust makespan from the POS generated from GNLA-guided local search. All runs on 4×4 and 6×6 instances took less than a minute, while 10×10 instances took about 15 minutes.

Table 4.1 provides the results. The performance of our solver is comparable to CB solver across all problem instances. This comparison illustrates that our local search mechanism is generic (different types of scheduling problems) and is also able to provide performance on par with near optimal approaches. While the performance is comparable, CB provides better MNPM values than our approach due to the following key reasons: (a) Our approach does not exploit the structure specific to JSPs (jobs consisting of a sequence of operations). We hope to improve our approach to exploit this in the near future. (b) Our robust local search reasons with upper bounds (due to Chebyshev inequality), which can be loose.

4.6 Application in Marine Industry

In the past decades, there has been a huge increase in the sea transportation industry, which leads a rapid growth in the ship MRO businesses. In Singapore, an annual turnover of close to 10 billion is generated and some 100,000 workers are employed in the marine industry, which plays a crucial part in Singapore's economic growth. These businesses on ship repair are competitive, in terms of not just price, but also time-to-delivery. A key requirement for a successful MRO business is that meeting schedules and completing work rapidly. In this section, we address a real RCPSP in the MRO industry in Singapore by using our proposed techniques. This is a joint work with A*STAR Singapore Institute of Manufacturing Technology.

This problem involves assigning tasks to a set of resources with limited capacity to minimize project makespan. The tasks from the data set are Set Keel Blocks into Position, Tow

	$\varepsilon = 0.5$	$\varepsilon = 0.2$	$\varepsilon = 0.1$	$\varepsilon = 0.05$	$\varepsilon = 0.025$
$\sigma = 0.1$	16.37	16.75	17.13	17.64	18.36
$\sigma = 0.2$	16.75	17.51	18.26	19.25	20.72
$\sigma = 0.5$	17.89	19.78	21.67	24.24	27.81

Table 4.2: Robust Makespan Varies with σ and ε.

vessel into drydock, Empty drydock, Undocking, High Pressure Washing, Grit Blasting, Painting, Renew zinc anodes, Remove/Cut/Weld Starboardside Sheet Plates, Empty Ballast Waste and Replace Ballast Water. The limited resources are Dock Master, Drydock, Dock Engineer, Dock Master, Rigger, Pilot Tugs, Fitter, Steel Cutter, Cutter, Welder and General Worker. To introduce durational uncertainty, we model the stochastic duration of each task as the expected duration set from given data plus an uncertain part following an normal distribution $N(0, \sigma)$.

Given uncertain durations of tasks and precedence relationship between tasks, resource consumptions and capacities, and the level of risk denoted as ε prescribed by the planner, the objective is to output an execution strategy with robust makespan $T*$, such that the probability of realized makespan for any schedule (derived from the execution strategy) does not exceed $T*$ is at least $(1 - \varepsilon)$, over all realizations of uncertainty. From our experiments, when no uncertainty is involved, the optimal makespan is 16 days; With durational uncertainty, Table 4.2 shows how robust makespan $T*$ varies with different sigma and epsilon. It provides the planner candidate execution strategies according to his own attitude to risk.

4.7 Conclusion

Given a level of risk $0 < \varepsilon \leq 1$ chosen by the planner, we investigated the problem of finding the minimum $(1 - \varepsilon)$-guaranteed makespan (i.e. Robust Makespan) and proposed methods to find a schedule policy (POS) such that when uncertainty is dynamically realized, the execution policy will result in a solution whose value is as good as robust makespan. We first put forward a new decision rule utilized in scheduling to help specify the start times for all activities with respect to execution policy and dynamic realizations of data uncertainty. Based on the decision rule, new fitness function was then derived to evaluate robustness, which was finally integrated into a local search method to produce the solution with robust makespan. Experimental results

illustrate the improved performance of local search with the new fitness evaluation, which provider tighter bounds on robust makespan and better partial order schedules compared to the existing method. In the end, we applied our model to the MRO industry context and provide the planner guidance of execution strategies based on his own attitude to risk.

For simplicity we have adopted an upper bound approach where we assume independence among the durational uncertainties. One future work is to treat correlations between durational uncertainties, since a task duration could be correlated with some others in real life. For example, correlations occur when an external event is not peculiar to a single task, but more universal, such as weather conditions, seasonal peaks. In such situations, the durational delays are correlated in the same direction. When this occurs, the decision rules proposed in this work break down unfortunately, since even if the covariances of pairs of duration variables are given, it is very complex to analytically model the extent to which one duration and any combination (resulting from *SUM* and *MAX* operators) of other durations change together. This in turn complicates the analysis on the variance of the makespan variable, and hence the robust makespan. Extending our work to handle covariances is an interesting future direction.

Chapter 5

Robust Execution Strategy under Resource and Durational Uncertainties

5.1 Contributions and Structure

Following on the work in Chapter 4, in this Chapter, we take into additional consideration of imprecise capacities of resources caused by breakdowns. Our objective is to provide robust execution strategy for RCPSP/max problems when both the assumptions of deterministic durations and fixed resource availabilities are relaxed. More specifically, the main structure and key contributions of this Chapter are given as follows:

- Firstly, we provide the motivation to study the scheduling problem with both resource and durational uncertainties. Building on an existing mechanism for solving RCPSP/max with durational uncertainty in [38], we evaluate the impact of resource breakdown on an activity's duration in Section 5.4;
- We then propose Forward-Backward Chaining for generating execution strategies in Section 5.5.1 with consideration of resource breakdown parameters. To improve robustness of resulting strategies, we further propose Resource-Breakdown-Aware Chaining procedure with three different metrics in Section 5.5.2. These chaining procedures computes a resource allocation by predicting the effect of breakdowns on robustness of the generated strategies.

- Finally, we extend the robust local search method proposed in [38] with additional considerations on the impact of resource breakdowns, so that generated execution strategies can better absorb resource and durational uncertainties.

- Experimental results demonstrate improvements in obtaining more robust execution strategies in Section 5.7, followed by the summation in Section 5.8.

5.2 Motivation

This work is motivated by the need to account for various kinds of disruptions in project scheduling. In managing disruptions for project scheduling, we need to handle three categories of disruptions [62]:

- Project network disruptions. It is possible that activities may be added or removed, or precedence relations are revised during execution.

- Activity disruptions. For example, activity duration varies since unexpected events occur during execution.

- Resource disruptions. That can be caused by factors like machine breakdown.

However, interruption does not usually occurs by an isolate factor, but often due to other disturbances [19]. In this work, we study both activity disruptions and resource disruptions.

In Chapter 4, we consider the objective of minimizing robust makespan for the case of the generalized project scheduling problem, RCPSP/max with stochastic activity durations. It is necessary and meaningful to deal with uncertain activity duration as many uncertain events, such as weather changes, productivity variability, etc. can affect activity duration during implementation. As a matter of fact, a large number of works on project scheduling under uncertainty deal with activity durational variability, for example, [5, 52, 41, 20, 16, 55].

In the bad environment conditions, however, the unavailability of resources is also an uncertain factor. As stated in [62], resource breakdowns is one of the most important sources of disruptions in project management. The primary impact of resource disruptions is schedule

infeasibility. However, to the best of our knowledge, there are not as much work on project scheduling with resource unavailability. [34, 36, 37, 35] solved RCPSP that represents a special case of RCPSP/max, with resource disruptions due to breakdowns. They worked for the robust schedule where robustness is measured by the weighted deviation between the planned and the actually realized activity starting times.

Following on the work in Chapter 4, we solve RCPSP/max with BOTH durational and resource uncertainties in this chapter. The motivation is that activities with stochastic durations should be considered, and at the same time, the need for dealing with unavailability of resource capacities due to unexpected breakdowns is very great for project schedule as well. To the best of our knowledge, this is the first work that deals with variable durations, resource breakdowns, and minimum/maximum time lags simultaneously in project scheduling. More precisely, we search for robust execution strategy with the best robust makespan. Given a level of risk $0 < \varepsilon \leq 1$, the robust makespan is a value for which the probability of realized makespan for any schedule derived from the execution startegy does not exceed it is greater than $(1 - \varepsilon)$, over all realizations of both resource and durational uncertainties. To obtain robust execution strategy, we propose Forward-Backward-Chaining algorithm and Resource-Breakdown-Aware Chaining procedures. These chaining procedures compute a resource allocation by predicting the effect of breakdowns on robustness of the generated strategies. Finally, we apply a local search method and compute robust execution strategy that absorb both resource and durational uncertainties.

5.3 Comparison with Existing Works on Resource Uncertainty

In the literature on project scheduling, there is not much work dealing with resource uncertainty for resource constrained project scheduling problems. [34, 36, 37, 35] considered resource uncertainty modeled by means of resource availabilities subject to unforeseen breakdowns for RCPSP, which is a special case of RCPSP/max.

[34, 36, 35, 37] represent a body of work that considers what is known as the instability

cost (or schedule nervousness) for the case of project scheduling with stochastic resource availabilities, which is defined as the sum of the weighted deviations between the original baseline schedule and the expected actual realized schedule. The instability weight reflects that activity's importance of being started as planned. This contrasts with our work, which aims at minimizing the Robust Makespan that provides a proven probability of successful execution under both resource and durational uncertainty.

[36, 37] focused on generating a predictive schedule by allocating time buffer into an initial, unbuffered schedule to increase robustness by absorbing the impact of resource breakdowns without rescheduling during execution. Instead of one schedule, we construct a POS that represents a set of feasible schedules and hence is more feasible than a baseline schedule. An attractive property of POS is fast response to unexpected disruptions; a new schedule can be quickly generated from the POS through a temporal propagation mechanism. Different from the analytical evaluation of resource breakdown in [37], we also compute the variance of durational extension due to resource breakdowns. The preempt-resume setting in [37] would be an interesting topic for our further research.

[34] described exact and suboptimal reactive scheduling procedures to revert online to a feasible schedule, that respects the new constraints created by the disruption and deviates as little as possible from the baseline schedule, so that the schedule infeasibility caused by the disturbances can be resolved. In our work, we solve the project scheduling problems under uncertainty from proactively. In [35], different proactive strategies were first proposed to generate robust baseline schedule with or without resource or time buffers. A list scheduling based reactive strategy or a tabu search based improvement heuristic can then be applied to indicate how to revert to a feasible schedule that deviates as little as possible from the original baseline. In [35], the interrupted activity has to be restarted from scratch very time it is interrupted, while in our work, we assume a preempt-resume setting. A possible extension of our work is to combine the online techniques proposed in [34] and [35] with our methods, thereby allowing us to implement proven robust execution strategies and good online behaviors.

5.4 Resource Breakdowns in RCPSP/max

In this section, we first provide the representation of resource uncertainty due to unexpected resource breakdowns and then analytically evaluate the effect of resource breakdown on activity durations.

5.4.1 Representation of Resource Breakdowns

Similar to existing research on resource constrained scheduling [37], we assume resources are renewable. Our approaches will compute a resource allocation for activities *before* project execution and this resource assignment remains fixed until the project finishes.

For a resource unit *j* of resource type *k*, we model the time between failures as a random variable X_{jk} and the time needed to repair this unit as a random variable Y_{jk}. Suppose that both X_{jk} and Y_{jk} are exponentially distributed. Then the corresponding cumulative distributions $F_{X_{jk}}(x)$ and $G_{Y_{jk}}(y)$ are given by:

$$\begin{aligned} F_{X_{jk}}(x) &= 1 - e^{-\lambda_{jk}x} \\ G_{Y_{jk}}(y) &= 1 - e^{-\mu_{jk}y} \end{aligned} \tag{5.1}$$

where $x, y \geq 0$, $1/\lambda_{jk}$ and $1/\mu_{jk}$ are the expected value of X_{jk} and Y_{jk}, respectively.

5.4.2 Analytical Effect of Resource Breakdown

Our approach in this work is to increase the durations of activities to account for resource breakdowns. Extending on existing research [37], we will now analytically measure the impact of resource breakdowns on an activity's duration. Given that we have a fixed resource assignment to each activity, we can precisely translate a resource breakdown as delay in the activity it has been assigned. Note that since we also consider the activity to already have an innate duration variability, this adds further variability to the activity. (We will consider both in section 5.6). Where we depart from [37] in our analysis is that we consider a more general case, i.e., even in the same resource type, we allow different mean time to failures and repair times for different resource units. Furthermore, for computing robust makespan, we also need to measure the

variance in addition to the mean values of the resulting duration variability.

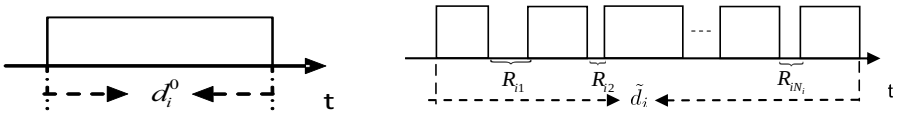
We first assume that resource breakdowns can only occur during activity execution. For simplicity, we assume only one resource unit can break down at any one time. Whenever a resource unit breaks down, it will be immediately sent for repair before it becomes available again. In the meantime, the activity corresponding to the resource unit is interrupted and has to wait for the resource to be repaired. It can resume from the point where execution was interrupted. Note that throughout the lifetime of an activity, there may be multiple interruptions and we assume that any two interruptions are independent from each other.

We use the following notation in this section:

- $\tilde{d}_i$: random variable representing the real duration of activity a_i under interruptions due to resource breakdowns
- d_i^0: deterministic duration of activity a_i when no interruption occurs
- δ_i: random variable representing total delay time of activity a_i due to resource breakdowns
- N_i: random variable representing number of interruptions due to resource breakdowns throughout the lifetime of execution for activity a_i
- T_i: random variable representing the minimum time to failure over all resource units allocated to activity a_i
- R_{ij}: random variable representing the time that activity a_i waits before it resumes (i.e. the repair time of the corresponding failed resource) at the j^{th} interruption
- X_{jk}: random variable representing the time to failure for the resource unit *j* of resource type *k*, $X_{jk} \sim EXP(\lambda_{jk})$
- Y_{jk}: random variable representing the repair time for the resource unit *j* of resource type *k*, $Y_{jk} \sim EXP(\mu_{jk})$
- M_k: random variable representing the minimum time to breakdown of all resource units of resource type *k* needed by activity a_i, $M_k = minX_{1k}, X_{2k}, ...X_{r_ik}$

We first model the duration variability of an activity a_i due to resource breakdowns as:

$$\tilde{d}_i = d_i^0 + \tilde{\delta}_i \tag{5.2}$$



(a) In deterministic environment. (b) Interrupted by resource breakdown.

Figure 5.1: Activity Duration.

The duration of activity a_i in the deterministic as opposed to an uncertain environment (due to resource breakdowns) are shown pictorially in Figure 5.1. According to definitions, we have the following relationships:

$$\tilde{\delta}_i = \Sigma_{j=0}^{N_i} R_{ij} \tag{5.3}$$

where R_{i0} equals to zero, which represents the case that no resource breakdown occurs during the execution of activity a_i. For simplicity, due to the independence relationship we assumed before, we henceforth rewrite R_{ij} as R_i. Then, from equation 5.3, we can calculate the mean value of $\tilde{\delta}_i$ as follows:

$$\begin{aligned} E(\tilde{\delta}_i) &= \textstyle\sum_{j=0}^{\infty} j * E(R_i) * P(N_i = j) \\ &= E(R_i) * E(N_i) \end{aligned} \tag{5.4}$$

To compute the variance of $\tilde{\delta}_i$ we need the following Lemma:

Lemma 5.4.1. *[51] Let $X_1, X_2, ...X_N$ be independent and identically distributed random variables with the same mean $E(X)$ and variance $Var(X)$. The sum has the type*

$$T = \textstyle\sum_{i=1}^{N} X_i$$

where N *is a random variable with a finite mean and variance and X_i are independent of* N*. Then the variance value of* T *is*

$$Var(T) = [E(X)]^2 Var(N) + E(N) Var(X).$$

From Lemma 5.4.1, the variance of $\tilde{\delta_i}$ can be calculated as:

$$Var(\tilde{\delta_i}) = [E(R_i)]^2 Var(N_i) + E(N_i)Var(R_i). \tag{5.5}$$

Hence, to determine the values of $E(\tilde{\delta_i})$ and $Var(\tilde{\delta_i})$, we need to calculate the mean and variance values of N_i and R_i. The detailed mathematical derivation will be shown in the following subsections.

Number of Interruptions.

In this subsection, we analyze the random variable N_i describing the total number of interruptions due to resource breakdowns experienced by activity throughout its execution.

Lemma 5.4.2. *[27] Let $X_1, X_2, ...X_n$ be independent exponential random variables with parameters $\lambda_1, \lambda_2, ...\lambda_n$, respectively. Then the minimum of these random variables follows an exponential distribution; that is,*

$$min\{X_1, X_2, ...X_n\} \sim EXP(\textstyle\sum_{i=1}^{n} \lambda_i).$$

During execution, activity a_i is interrupted as soon as one of these resource units allocated to a_i breaks down, and thus, T_i describing the time to failure of activity a_i can be shown as the minimum time to failure over all of these resource units:

$$T_i = min\{X_{11}, X_{21}, ..., X_{r_{i1}1}, ..., X_{1K}, X_{2K}, ..., X_{r_{iK}K}\}. \tag{5.6}$$

We then rewrite T_i as:

$$T_i = min\{M_1, M_2, ..., M_K\} \tag{5.7}$$

where M_k represents the minimum time to failure of resource type *k* allocated to activity a_i.

Since $X_{jk} \sim EXP(\lambda_{jk})$, using Lemma 5.4.2, we then have,

$$M_k \sim EXP(\textstyle\sum_{j=1}^{r_{ik}} \lambda_{jk}) \tag{5.8}$$

Similarly, following equations 5.7, 5.8 and Lemma 5.4.2, we can see that T_i is also exponentially distributed with the parameter $\sum_{k=1}^{K} \sum_{j=1}^{r_{ik}} \lambda_{jk}$, so that

$$E(T_i) = 1/\sum_{k=1}^{K} \sum_{j=1}^{r_{ik}} \lambda_{jk} \tag{5.9}$$

Lemma 5.4.3. *[27] Suppose the time between consecutive occurrences of arrivals follows an exponential distribution with parameter λ. Let* X(t) *be the number of occurrences by time* t*, then* X(t) *follows a Poisson distribution with parameter $\lambda * t$.*

Using Lemma 5.4.3, N_i representing the number of interruptions of activity a_i during its execution, follows a Poisson distribution with the mean number of occurrences given by $d_i^0/E(T_i)$: Consequently, the mean and variance values of N_i are both $d_i^0/E(T_i)$, i.e.

$$E(N_i) = Var(N_i) = d_i^0 \sum_{k=1}^{K} \sum_{j=1}^{r_{ik}} \lambda_{jk} \tag{5.10}$$

Total Vacancy Time.

Lemma 5.4.4. *[37] Let* X *and* Y *be independent random variables that are both exponentially distributed, respectively with parameter λ and μ. The probability that* X *will be smaller than* Y *is then:*

$$P(X < Y) = \lambda/(\lambda + \mu)$$

Let P_{jk} denote the probability that the interruption for activity a_i is caused by the resource unit *j* of resource type *k*. That is, among all resource units used by activity a_i, this resource unit has the smallest time to breakdown. Then P_{jk} can be calculated as:

$$P_{jk} = P(X_{jk} < min_{l=1,\ldots K; i=1,\ldots r_{il}; (i,l)\neq(j.k)} X_{il}) \tag{5.11}$$

From equation 5.11 and Lemma 5.4.4, we can rewrite:

$$P_{jk} = \frac{\lambda_{jk}}{\lambda_{jk} + \sum_{l=1,\ldots K; i=1,\ldots r_{il}; (i,l)\neq(j.k)} \lambda_{il}} = \frac{\lambda_{jk}}{\sum_{l=1}^{K} \sum_{i=1}^{r_{il}} \lambda_{il}} \tag{5.12}$$

Since we assume that only one unit of resource is allowed to break down at a time, we can

calculate the mean and variance values of R_i describing the vacancy time for a_i to wait once interrupted as:

$$E(R_i) = \sum_{k=1}^{K} \sum_{j=1}^{r_{ik}} P_{jk} E(Y_{jk})$$

$$\begin{aligned} Var(R_i) &= E(R_i^2) - E^2(R_i) \\ &= \sum_{k=1}^{K} \sum_{j=1}^{r_{ik}} P_{jk} E(Y_{jk}^2) - E^2(R_i) \\ &= \sum_{k=1}^{K} \sum_{j=1}^{r_{ik}} P_{jk} (Var(Y_{jk}) + E^2(Y_{jk})) - E^2(R_i) \end{aligned} \tag{5.13}$$

Here, the random variable Y_{jk} represents the time needed to repair for the resource unit *j* of resource type *k* once broken. Since we assume that $Y_{jk} \sim EXP(\mu_{jk})$, then we can obtain:

$$\begin{aligned} E(Y_{jk}) &= 1/\mu_{jk} \\ Var(Y_{jk}) &= 1/(\mu_{jk})^2 \end{aligned} \tag{5.14}$$

Therefore, from equations 5.4, 5.5, 5.10, 5.13 and 5.14, the mean and variance values of the stochastic part of duration can be derived as:

$$\begin{aligned} E(\tilde{\delta}_i) &= d_i^0 \sum_{k=1}^{K} \sum_{j=1}^{r_{ik}} \frac{\lambda_{jk}}{\mu_{jk}} \\ Var(\tilde{\delta}_i) &= d_i^0 \sum_{k=1}^{K} \sum_{j=1}^{r_{ik}} \frac{2\lambda_{jk}}{(\mu_{jk})^2} \end{aligned} \tag{5.15}$$

5.5 Chaining Heuristics

5.5.1 Forward-Backward-Chaining

In this section, we discuss a new chaining procedure Forward-Backward-Chaining (FBC) that produces a POS and a resource assignment that takes into account resource breakdowns.

Resource Ranking Heuristic

Given that the distributions of breakdown and repair time vary across resource units, it is clear that different choices of resource assignments to activities will have different effects on robustness. In this section, we ignore the effects of repair time and focus instead on the breakdown distribution instead. A desirable property is to have an optimal resource assignment to activi-

ties such that the maximal (or other metrics such as sum of) probability of breakdown over all activities is minimized. However, even such a simple case involving two activities on a given breakdown distribution can be proven to be NP-hard via a reduction from Partition Problem.

In the following, we propose a heuristic to establish a ranking among resource units according to their breakdown (or reliability), which will be used by the chaining procedure subsequently in assigning resources to activities.

Given two arbitrary resource units with the time to breakdown denoted as X and Y whose cumulative distribution are F and G, respectively. If at any time x, the probability of breakdown at or before time x is lower under F than under G, then intuitively X is a preferred resource choice over Y. This intuition leads us to apply the concept of Stochastic Dominance [39], which is a form of stochastic ordering used commonly to establish the order of preference between two random variables. The canonical form of stochastic dominance is the first-order dominance which we use in this work:

Definition. Let X and Y denote two random variables whose cumulative distributions are F and G, respectively. X stochastically dominates Y, denoted by XDY, if and only if $F(X) \leq G(X)$ for all x.

This stochastic dominance definition implies that if *XDY* then *F* must be below *G* for the whole range of *x*. Thus in our context, we can have the following proposition.

Proposition 4. *Let* X *and* Y *denote the time to breakdown of two resource units whose cumulative distributions are* F *and* G*, respectively. Assume that* X *and* Y*, the two random variables, are exponentially distributed with the following parameters:*

$$X \sim EXP(\lambda)$$
$$Y \sim EXP(\mu)$$

Then X *stochastically dominates* Y *(*XDY*), if and only if* $\lambda < \mu$.

Algorithm

Various chaining algorithms have been proposed in the literature (e.g. [49]) to generate a POS based on a baseline schedule. According to our experiments, the robust makespan is highly

correlated with the deterministic makespan. In the section, we propose a new iterative forward-backward chaining procedure which is capable of producing good deterministic makespans. To cope with resource breakdowns, this procedure also incorporates the greedy resource assignment heuristic based on the resource ranking described above.

Suppose that all resource units have been ranked according to non-increasing order of reliability. The following FBC procedure will return a *POS* and resource assignment *R*. The variable used in this function are all local variables.

Algorithm 3 FBC(*S*)

Improved ← True
calculate deterministic makespan *MK* based on *S*
set $V^* \leftarrow \infty$
while *Improved* **do**
  $(POS', R') \leftarrow$ Forward-Chaining(*S*)
  calculate robust makespan $V^{*'}$ based on (POS', R')
  if $V^{*'} < V^*$ **then**
    $POS \leftarrow POS', V^* \leftarrow V^{*'}, R \leftarrow R'$
  end if
  calculate new baseline schedule*S* based on POS'
  $(POS', R') \leftarrow$ Backward-Chaining(*S*)
  calculate robust makespan $V^{*'}$ based on (POS', R')
  if $V^{*'} < V^*$ **then**
    $POS \leftarrow POS', V^* \leftarrow V^{*'}, R \leftarrow R'$
  end if
  calculate new baseline schedule*S* based on POS'
  calculate deterministic makespan MK' based on *S*
  if $MK' < MK$ **then**
    Improved ← True
  else
    Improved ← False
  end if
end while
return (POS, R)

Let $chain_{kp}$ represent the chain associated with unit *p* of resource type *k*, and R_{ijk} store the index of the resource unit of resource type *k* that is assigned as the j^{th} resource unit to activity *i*. The detailed Forward-Chaining function which returns a *POS* and resource assignment *R* is given as follows,

Selectchain is a greedy resource assignment heuristic that selects the index *p* of the first

Algorithm 4 Forward-Chaining(*S*)

```
sort activities by start times in S, ties broken randomly
initialize all chains with dummy activity a_0
for i = 1 to N do
    for k = 1 to K do
        for j = 1 to r_ik do
            p ← Selectchain(a_i, r_k)
            a_l ← last(chain_kp)
            R_ijk ← p
            add precedence constraints between a_l and a_i to POS
        end for
    end for
end for
return (POS,R)
```

available chain of resource type k where the ending time of the last activity a_l on the chain is earlier than the start time of activity a_i. Since the resource units are already sorted in non-increasing order of reliability, it will thus pick the most stochastically reliable resource units to be assigned to the activity. Backward-Chaining is similar to Forward-Chaining except the activities are instead sorted according to finish time and start chaining backward. Note that similar to the Forward-Backward Improvement (FBI) algorithm of [32] for RCPSP, each call of the forward or backward chaining function will only result in a better or equal makespan.

5.5.2 Resource-Breakdown-Aware Chaining

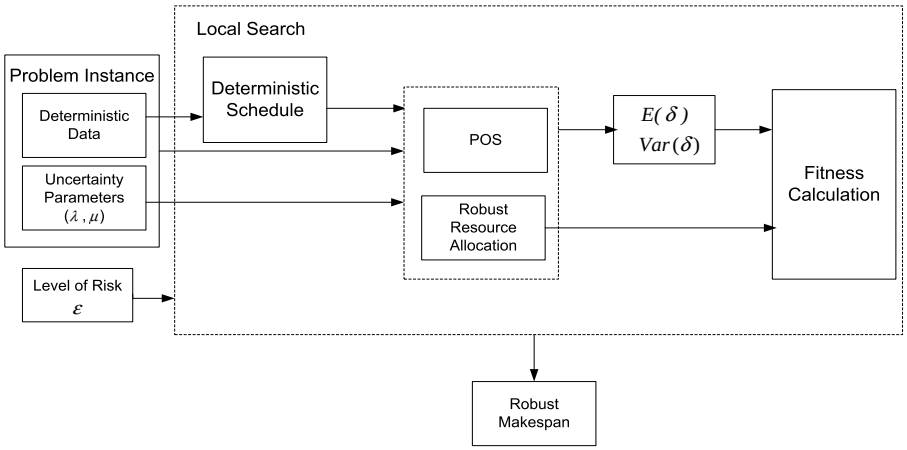


Figure 5.2: Overview of Local Search.

We provide a simplified overview of our robust local search in Fig 5.2 to motivate the need

for resource assignment heuristics in this section. Typically, there exist multiple feasible resource assignments for every activity during the chaining procedure (for constructing a POS). It should be noted that different assignments can lead to different execution strategies (or POSs), each with a different level of robustness (or equivalently robust makespan). In Section 5.5.1, resources are firstly ranked based on the mean value of the time to breakdown (λ) and activities are greedily allocated to the first available resource based on that order. However, such a greedy assignment is not sensitive to resource breakdown and repair distributions. Therefore, we build on the latest chaining technique for POS [47] by proposing heuristics for assigning activities to resource chains (or units) based on the breakdown (λ) and repair (μ) distribution parameters.

A key observation in this regard is the dependence of Equation 5.21 on the terms $\frac{\lambda}{\mu}$ and $\frac{\lambda}{\mu^2}$. This implies that expected value and variance of robust makespan are dependent directly on uncertainty parameters relevant to resource breakdowns. Based on this observation, we propose our chaining algorithm called Resource Breakdown Aware Chaining. Intuitively, we will assign longer duration activities to more reliable resources, where the reliability is characterized by either of the three heuristics:

- Mean heuristic (MH), where reliability is defined by $\frac{\lambda}{\mu}$
- Variance heuristic (VH), where reliability defined by $\frac{\lambda}{\mu^2}$
- Mean and Variance heuristic (MVH), where reliability is a combination of $\frac{\lambda}{\mu}$ and $\frac{\lambda}{\mu^2}$.

An analytical proof of the effectiveness of MVH for a simple two activity case is provided as follows.

Example 6. *Consider two activities that are to be executed in parallel (within a POS). Duration for an activity i is represented as $\tilde{d}_i = d_i^0 + \tilde{l}_i - \tilde{e}_i + \tilde{\delta}_i$, where $i = 1, 2$ and $d_1^0 > d_2^0$. Furthermore for simplicity, let us assume two units of one resource type and the resource consumption for each activity is one. unit. We assume that further that $\lambda_1/\mu_1 < \lambda_2/\mu_2$ and $\lambda_1/\mu_1^2 < \lambda_2/\mu_2^2$.*

According to the decision rule SLA, the makespan for the two activity network is upper bounded by: $max\{d_1^0, d_2^0\} + \tilde{l}_1 + \tilde{l}_2 + \tilde{\delta}_1 + \tilde{\delta}_2$. We have two resource assignments:

- *Assignment 1: assign activity 1 to resource unit 1 and activity 2 to resource unit 2*

- *Assignment 2: assign activity 2 to resource unit 1 and activity 1 to resource unit 2.*

We now show that assigning activities according to MVH can provide lower robust makespan. The values of robust makespan following assignment 1 and 2 are denoted as S_1^* *and* S_2^**, respectively. We then have,*

$$\begin{aligned} S_1^* &= max\{d_1^0, d_2^0\} + E(\tilde{l}_1) + E(\tilde{l}_2) + d_1^0\frac{\lambda_1}{\mu_1} + d_2^0\frac{\lambda_2}{\mu_2} + \\ &\quad \sqrt{\frac{1-\varepsilon}{\varepsilon}}\sqrt{Var(\tilde{l}_1) + Var(\tilde{l}_2) + d_1^0\frac{2\lambda_1}{\mu_1^2} + d_2^0\frac{2\lambda_2}{\mu_2^2}} \\ S_2^* &= max\{d_1^0, d_2^0\} + E(\tilde{l}_1) + E(\tilde{l}_2) + d_1^0\frac{\lambda_2}{\mu_2} + d_2^0\frac{\lambda_1}{\mu_1} + \\ &\quad \sqrt{\frac{1-\varepsilon}{\varepsilon}}\sqrt{Var(\tilde{l}_1) + Var(\tilde{l}_2) + d_1^0\frac{2\lambda_2}{\mu_2^2} + d_2^0\frac{2\lambda_1}{\mu_1^2}} \end{aligned} \tag{5.16}$$

Given $d_1^0 > d_2^0$*,* $\lambda_1/\mu_1 < \lambda_2/\mu_2$ *and* $\lambda_1/\mu_1^2 < \lambda_2/\mu_2^2$*, we have,*

$$\begin{aligned} &d_1^0\tfrac{\lambda_1}{\mu_1} + d_2^0\tfrac{\lambda_2}{\mu_2} < d_1^0\tfrac{\lambda_2}{\mu_2} + d_2^0\tfrac{\lambda_1}{\mu_1}, \\ &d_1^0\tfrac{2\lambda_1}{\mu_1^2} + d_2^0\tfrac{2\lambda_2}{\mu_2^2} < d_1^0\tfrac{2\lambda_2}{\mu_2^2} + d_2^0\tfrac{2\lambda_1}{\mu_1^2} \end{aligned} \tag{5.17}$$

Thus, $S_1^* < S_2^*$.

Algorithm 5 provides the pseudo code for resource breakdown aware chaining. Line 4 - Line 8 and Line 20 - Line 29 constitute the chaining technique [47] introduced in Section 3.2.1 that aims to improve the flexibility (or parallelism). We improve this chaining procedure to incorporate sensitivity to resource breakdowns in lines 9-18. Depending on the exact heuristic employed, the last activity in a resource chain (or unit) is swapped so that a longer duration activity is scheduled on a more robust resource.

5.6 Extended Robust Local Search

We now extend the robust local search method with additional considerations on the impact of resource breakdowns to the robust makespan of a given POS. First, we revise the segregated linear decision rules introduced in Section 4.2. The idea is to be able to separate the effects of

Algorithm 5 Resource Breakdown Aware Chaining, RBAC (Activity a, Schedule S, Heuristic h)

$C(a) \leftarrow$ Find set of available chains, $C(a)$ for activity a based on S
$U(a) \leftarrow$ Find set of unavailable chains, $U(a)$ for activity a based on S
$P(a) \leftarrow$ Collect chains from $C(a)$ with last activity of chain preceding a in problem
if $P(a) \neq \phi$ **then**
    $k \leftarrow$ Get an available chain in $P(a)$
else
    $k \leftarrow$ Get an available chain in $C(a)$
end if
Let b denote the last activity of chain u
if $\exists\ u \in U(a)$, $k \in C(b)$,$dur_a > dur_b$ **then**
    if h = "**MH**", $\lambda_k/\mu_k > \lambda_u/\mu_u$ **then**
        swap activity b onto chain k, $k \leftarrow u$
    else if h = "**VH**", $\lambda_k/\mu_k^2 > \lambda_u/\mu_u^2$ **then**
        swap activity b onto chain k, $k \leftarrow u$
    else if h = "**MVH**",$\lambda_k/\mu_k > \lambda_u/\mu_u$, $\lambda_k/\mu_k^2 > \lambda_u/\mu_u^2$ **then**
        swap activity b onto chain k, $k \leftarrow u$
    end if
end if
Post constraint between between last activity of chain k (denoted as $last(k)$) and activity a

if a requires more than one resource unit **then**
    $C1(a) \leftarrow$ chains in $C(a)$ which have last activity as last(k)
    $C2(a) \leftarrow C(a) \setminus C1(a)$
    for all resource units required by a **do**
        choose an available chain belonging to $C1(a)$
        if chain above is not feasible **then**
            choose an available chain belonging to $C2(a)$
        end if
    end for
end if

the innate duration variability and the delay due to resource breakdown. For this purpose, we introduce two notions: a real activity and a dummy activity. For each activity a_i, we divide it into a "real" activity a_i' and a "dummy" activity a_i'' as illustrated in Figure 5.3. Thus, duration

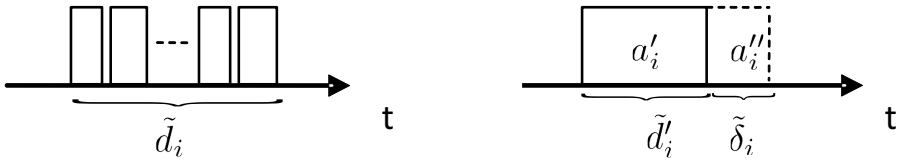


Figure 5.3: "Real" activity and "Dummy" activity.

of an activity is a sum of duration of its real part and the duration of the dummy part. i.e. $\tilde{d}_i = \tilde{d}_i' + \tilde{\delta}_i$, where the duration of its dummy activity is modeled as $\tilde{\delta}_i$. We then model the duration of the real part as:

$$\tilde{d}_i' = d_i^0 + \tilde{z}_i = d_i^0 + \tilde{l}_i - \tilde{e}_i \tag{5.18}$$

where the segregated random variables $\tilde{l}$ and $\tilde{e}$ represent the distributions of the lateness and earliness, respectively.

In a scheduling context, an activity will either start after the end of an activity (i.e. in series) or after the end of multiple activities occurring simultaneously (i.e. in parallel). Now we describe the computation of $\tilde{S}_v'(\mathbf{x}, \tilde{\mathbf{z}}, \tilde{\delta})$ (compared with $\tilde{S}_v(\mathbf{x}, \tilde{\mathbf{z}})$ under only durational uncertainty in Section 4.2) by representing both resource and durational uncertainties for activities:

- Serial Activities: The start time of the activity starting after the k-activity project is expressed as:

  $$\tilde{S}_k'(\mathbf{x}, \tilde{\mathbf{z}}, \tilde{\delta}) = \textstyle\sum_{i=1}^{k}(d_i^0 + \tilde{\delta}_i + \tilde{z}_i^+ - \tilde{z}_i^-). \tag{5.19}$$

  Mean and variance of the segregated variables $\tilde{z}^+$ and $\tilde{z}^-$ are known, and mean and variance of variation $\tilde{\delta}_i$ can be computed according to Eqn 5.15, hence the mean and variance of $\tilde{S}_k'$ can be easily computed.

- Parallel Activities: Consider activities that are executed concurrently, the upper bound on the start time of an activity starting after the parallel k-activity project network is

represented as follows:

$$\tilde{S}'_k(\mathbf{x}, \tilde{\mathbf{z}}, \tilde{\delta}) \leq \max_{i=1,\ldots k}\{d_i^0\} + \sum_{i=1}^{k} \tilde{z}_i^+ + \sum_{i=1}^{k} \tilde{\delta}_i. \tag{5.20}$$

Similarly, mean and variance of the segregated variables are known, and mean and variance of variation $\tilde{\delta}_i$ can be computed according to Eqn 5.15, hence the mean and variance values of upper bound of $\tilde{S}'_k$ are easy to compute.

Based on the segregated linear decision rule under both resource and durational uncertainties, the robust fitness function which will be used in local search can defined as follows:

Definition 4. *Given* $0 < \epsilon \leq 1$ *and the adjustable fitness function* $\tilde{S}'(\mathbf{x}, \tilde{\mathbf{z}}, \tilde{\delta})$ *defined above, the robust fitness function,* $f'(x, \tilde{z}, \tilde{\delta}, \epsilon)$*, is defined as*

$$f'(\mathbf{x}, \tilde{\mathbf{z}}, \tilde{\delta}, \epsilon) = E[\tilde{S}(\mathbf{x}, \tilde{\mathbf{z}}, \tilde{\delta})] + \sqrt{\frac{1-\epsilon}{\epsilon}}\sqrt{Var[\tilde{S}(\mathbf{x}, \tilde{\mathbf{z}}, \tilde{\delta})]} \tag{5.21}$$

We are now ready to present our improved Robust Local Search which is capable of handling resource and durational uncertainties. The goal of the local search mechanism is to find a local minima of robust makespan S'^*. The detailed description of our Robust Local Search with Resource Breakdown Aware Chaining and MVH heuristic (RLS_{MVH}) is presented as follows in Algorithm 6.

5.7 Experimental Evaluation

In this section, we compare the quality of the execution strategies generated by our robust local search approach with the new chaining algorithms against robust local search with existing chaining algorithms.

5.7.1 Experimental Setup

The problems considered for RCPSP/max with resource and durational uncertainties were obtained by extending the three benchmark sets available for RCPSP/max problems, J10, J20 and

Algorithm 6 RLS_{MVH}

rank resource units in non-increasing order of reliability (by stochastic dominance)
initialize activity list *AL*
set $i \leftarrow 0; S'^{*} \leftarrow \infty$
while $i < Max_Iteration$ **do**
  generate baseline schedule *S* according to *AL*
  initialize the POS
  for all $i \leq N$ **do**
    call RBAC(a_i, *S*, "MVH") to add to (*POS,R*)
  end for
  calculate robust makespan *V* according to (*POS,R*)
  if $S < S'^{*}$ **then**
    set $S'^{*} \leftarrow S$,and update *AL* and *R*
  end if
  $i \leftarrow i + 1$
  apply local move on *AL* to generate new *AL*
end while
return S'^{*}

J30 as specified in the PSPLib [31]. Each set contains 270 problem instances with duration for each activity ranging between 1 and 10. The maximum number of activities for J10, J20 and J30 are 10, 20 and 30, respectively. For each activity a_i, we set the fixed part d_i^0 of the stochastic duration as the corresponding deterministic duration given by the benchmarks, and assume that duration uncertainty is normally distributed, i.e. $\tilde{z}_i \sim N(0, \sigma)$. We set $\sigma = 0.1$ and run the algorithms on four increasing levels of risk $\varepsilon = \{0.01, 0.05, 0.1, 0.2\}$. As for the resource uncertainty, we set λ and μ each randomly picked from one of the intervals, respectively: $1/\lambda \in \{[5, 15], [15, 25], [25, 35]\}$, $\mu \in \{[1, 2], [2, 3], [3, 4]\}$. Intuitively, these represent a decent range of values for number of breakdowns and the time for repair.

We compare the robust local search with different chaining procedures: a) (RLS_{MH}) is the robust local search with mean heuristic chaining ; b) (RLS_{VH}) is the robust local search with variance heuristic chaining; c) (RLS_{MVH}) is the robust local search with combined mean variance chaining; d) (RLS_{FBC}) refers to the robust local search with Forward-Backward Chaining; e) (RLS) refers to the robust local search with random Chaining.

The number of local search iterations for robust local search was set to 1000. To reduce the stochasticity effects of robust local search, we average over 10 random executions for each problem instance. Our code was implemented in C++ and executed on a Core(TM)2 Duo CPU

Set	Algorithm	$\Delta_{LB}\%$	$N_{Opt}\%$	$N_{Feas}\%$	S^*	CPU
J10	RLS_{FBC}	0.00	100	100	46.3	0.20
	RLS	0.16	93.6	100	46.6	0.14
J20	RLS_{FBC}	4.83	70.1	100	75.2	0.92
	RLS	9.35	40.1	100	78.3	1.20
J30	RLS_{FBC}	12.31	47.6	100	98.4	2.30
	RLS	19.63	20.54	99.5	103.8	4.87

Table 5.1: Effectiveness of RLS_{FBC} on Benchmark Problems.

2.33GHz processor under FedoraCore 11 (Kernel Linux 2.6.29.4-167.fc11.i586).

5.7.2 Comparisons between Chaining Procedures

To test the performance of our proposed chaining procedures, we compare the average robust makespans of 270 problem instances under different levels of risk.

In Table 5.1, we compare the performance of RLS_{FBC} with ***RLS*** where ε is set to 0.05. We observe that not only is a better robust makespan achieved for all the test sets by RLS_{FBC}, but also our approach obtains much better results in terms of deviation from lower bound ($\Delta_{LB}\%$), percent of optimal solution found ($N_{Opt}\%$). RLS_{FBC} is also faster computationally than ***RLSL*** on the J20 and J30 test sets. We also like to remark that RLS_{FBC} and their algorithm are the only algorithms to our knowledge which can find all feasible solutions for J10, J20 and J30 test problems.

Table 5.2 shows how different chaining procedures and risk levels ε affect the robust makespan over different data sets, each of which are generated randomly with $1/\mu$ and λ set to [5,15] and [1,2], respectively. While we considered one set of values for $1/\mu$ and $lambda$, we observed similar results for other range of values. Here are the key conclusions:

- Irrespective of problem scale, as the level of risk ε increases, the robust makespan decreases with all four algorithms. This is an expected result, where the lower risk that the scheduler is willing to take, the higher is the robust value of the generated execution strategy;

- Irrespective of problem scale and level of risk, our proposed methods (RLS_{MH},RLS_{VH} and RLS_{MVH}) can provide better robust makespan than [21]. This indicates that consid-

Problem Sets	Algorithms	$\varepsilon = 0.01$	$\varepsilon = 0.05$	$\varepsilon = 0.1$	$\varepsilon = 0.2$
J10	RLS_{MH}	87.37	91.74	97.69	122.14
	RLS_{VH}	87.11	91.55	97.59	122.41
	RLS_{MVH}	86.70	91.12	97.11	121.75
	RLS_{FBC}	90.13	94.60	100.68	125.67
J20	RLS_{MH}	280.27	286.48	294.91	329.61
	RLS_{VH}	273.27	279.41	287.76	322.06
	RLS_{MVH}	263.92	270.02	278.31	312.37
	RLS_{FBC}	328.68	334.93	343.42	378.27
J30	RLS_{MH}	327.73	335.15	345.23	386.69
	RLS_{VH}	336.56	343.91	353.89	394.97
	RLS_{MVH}	318.29	325.58	335.50	376.27
	RLS_{FBC}	353.58	360.97	371.01	412.21

Table 5.2: Comparison of RLS_{MH}, RLS_{VH}, RLS_{MVH} against RLS_{FBC} on Benchmark Problems.

ering robustness chaining in improving robustness of generated execution strategy;

- Irrespective of problem scale and level of risk, RLS_{MVH} performs better than $RLS_{MH}$and RLS_{VH}, but we do not see any superiority between RLS_{MH} and RLS_{VH}.

Effect of Breakdown Parameters on Robustness

We now show the effect of different breakdown parameters on robustness of generated execution strategies. We consider all the values of μ and λ shown in Table5.3. While the risk level ε is set to 0.2 for these results, similar results were observed for other epsilon values as well.

The key conclusion is that the robust makespan increases when λ increases, and it decreases when μ increases. Since mean and variance parts of the resource uncertainty increase when λ increases according to equation 5.15, while the values decrease when μ increases.

5.8 Conclusion

In this work, we provided a scalable architecture for solving RCPSP/max problems with durational and resource breakdown uncertainties. The major contribution of the paper is the Resource Breakdown Aware Chaining procedure with three different metrics (MH, VH and MVH)

Problem Sets			$\mu \in$ *[1,2]*	$\mu \in$ *[2,3]*	$\mu \in$ *[3,4]*
J10	$1/\lambda \in$	*[5,15]*	86.70	64.61	57.73
		[15,25]	59.21	52.54	50.17
		[25,35]	54.18	49.92	48.30
J20	$1/\lambda \in$	*[5,15]*	263.92	112.96	96.48
		[15,25]	97.82	85.79	82.14
		[25,35]	88.54	81.23	78.83
J30	$1/\lambda \in$	*[5,15]*	318.29	201.99	133.19
		[15,25]	133.55	112.94	106.20
		[25,35]	116.78	105.91	102.52

Table 5.3: Robust Makespan with Different Breakdown Parameters.

to obtain execution strategies based on resource breakdown and repair distributions. Furthermore, we have provided formal mechanisms for (i) predicting the effect of resource breakdowns and repairs on robust makespan; (ii) suggesting more robust resource allocations. Experimental results illustrate improved robustness of execution strategies with the new chaining technique.

There are two key assumptions that we hope to relax in future work:

- Firstly, we have assumed exponential distributions for times between resource breakdowns and repair times once resources breakdown. An immediate relaxation of this assumption would be to address RCPSP/max with only mean and variance of resource uncertainty (rather than any specific distribution). Furthermore, the general non-linear decision rule we proposed in Section 4.2.1 that contributes for computing schedules with respect to execution strategy and realizations of uncertainty would also be applied for generating robust execution strategies.

- While we consider different resource allocations during the chaining procedure,the resource assignment will remain fixed during execution once it is generated. We hope to relax this using the online approaches contributed by [34].

Chapter 6

Application in Service Industry Context

6.1 Introduction

In service industry such as airlines, health-care and financial institutions, timely delivery of *product* (goods and services) is of utmost importance in gaining customers' confidence and goodwill. While JIT has been an established concept in lean manufacturing, it is also widely applicable in the service industry (where jobs should be completed to fulfill customers' orders in a timely fashion against a backdrop of dynamism and uncertainty in the environment). The notion of timeliness means that the jobs are completed neither too early (thereby minimizing holding cost and maximizing agility) nor too late (minimizing delays and customer waiting time).

In this chapter, we study the problem of timely service delivery in an operational environment that is inherently uncertain. The broad research question is - can we obtain a policy on apriori (proactive) resource assignment to jobs that is robust in response to events such as a demand surge? We seek a rigorous modeling of uncertainty parameters and a computationally efficient approach to find solutions that are robust against such uncertainties. More precisely, we will look at a job scheduling context where we need to find robust solutions that are JIT. In Scheduling terminology, the JIT concept is often studied as the well-known earliness/tardiness (E/T) scheduling problems. A rich collection of scheduling problems in the service industry is given in [58].

While there is a rich literature in dealing with E/T problems (mostly in manufacturing contexts), they mainly deal with deterministic problems on single-machine or parallel machines. Baker and Scudder [3] provided a comprehensive survey on the different variations of deterministic E/T problems. In a deterministic setting, it is meaningful to seek a schedule (an assignment of start time of each task to a machine) that optimizes an objective function (such as makespan). However, in a dynamic operational environment particularly in a service company where we need to cope with the presence of uncertainties realistically, a schedule is potentially brittle against uncertainty (such as durational uncertainty).

To hedge against uncertainty, a widely adopted technique known as the Critical Chain was proposed by Goldratt [23] for project management. The idea was derived from Theory of Constraints. A critical chain is the longest sequence of both precedence and resource-dependent tasks in a project, and the idea is to insert *project buffer* time to the end of a critical chain to protect the project completion date, as well as *feeding buffer* time that hedges against resource contention of a task on a non-critical chain with a task on a critical chain should the former gets delayed. While buffering is an intuitive notion that is fairly simple to implement, the question is how much to buffer. In a typical critical chain approach, the size of the buffer increases linearly with the length of the chain with which it is associated, and in [25], it was reported that the resulting makespan based on critical chain can be twice as long compared to the scheme proposed by the authors' approach.

In Operations Research, stochastic programming and simulation techniques are commonly used to handle uncertainty - both of which require high computational budget typically (for large-scale problems) and rely heavily on statistical distributions. These are overcome with techniques from Artificial Intelligence and Economics. A survey by Herroelen and Leus [26] review these fundamental approaches for scheduling under uncertainty. More recently, the notion of robust optimization has been proposed that makes use of mild statistical information to find tractable solutions for optimization problems under uncertainty [6].

A recent approach that combines robust optimization with heuristic search to manage uncertainty proactively is based on the idea of robust local search [38]. In applying to project scheduling with resource constraints, [21, 22] proposed computationally efficient schemes for finding an *executable strategy* (or policy) that hedges against durational and resource uncer-

tainties. More precisely, by making use of mild statistical information (means and variance) they proposed a computationally efficient scheme to generate a partial-order schedule (POS) (i.e. an execution policy) and a "robust" makespan guaranteeing that the schedule subsequently executed against all possible realizations of uncertainty will have a makespan of that value with a probability of $1 - \epsilon$, where $\epsilon \in [0, 1]$.

In this chapter, we adopt the idea proposed in the abovementioned works to study the problem of E/T scheduling under durational uncertainty. We term our problem as Dynamic E/T Job Scheduling Problem. We believe this problem, though somewhat stylized in nature, is capable of modeling many of problems arising from job assignment in the service industry. We see many real-life business process scenarios that are plagued with both E/T and duration uncertainty considerations. One such application is in the production and delivery of perishable goods, such as flight catering business. A job in flight catering business includes multiple operations such as cooking for hot dish, cold dish, preparation of beverages and special meals, assembly of items and packing. An early completion of job results in holding costs and possible inaccuracy but a late completion of job will have severe impact on flight schedule, loss of customer satisfaction and potential disruption to other flights (if food is *borrowed* from another flight). The study of uncertainty is important as passenger load (paxload) is dynamic due to ticket sales and check-ins. The airlines update the caterer on the paxload as late as within 40 minutes before the flights departure. As paxload changes, the durations required to perform some operations such as assembly and packing could vary.

In our work, we aim to find a schedule policy with the *least robust cost* of E/T under duration uncertainty. Our contribution in this work is three-fold. Firstly, under a stochastic setting, we provide an analytical method to evaluate the expected cost of E/T for each job as well as for all jobs for a given schedule. Secondly, we show that with the use of a executable strategy, we can gain greater visibility into a job completion time. Finally, we propose the use of an efficient local search method in search of a time-and-resource-feasible schedule that gives the minimum robust cost for all jobs in the project for a given risk level ϵ.

The organization of the rest of the work is as follows: Section 6.2 describes a business process of a service provider that motivates our work. Section 6.3 provides a formal definition of the problem. Section 6.4 provides the detailed description about our approach to solve the

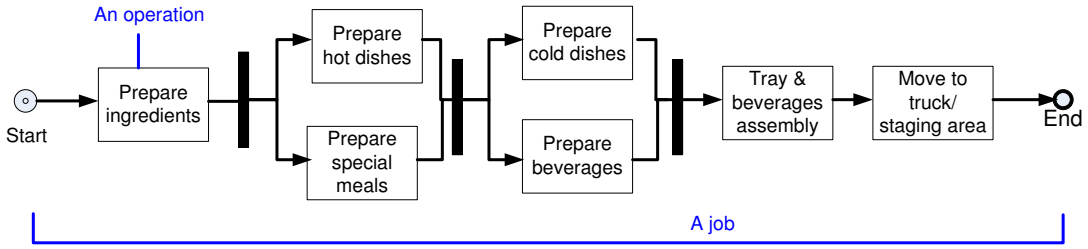


Figure 6.1: Flight catering business process as an example of a job

problem. Section 6.5 shows the experimental results for our method. Finally, Section 6.6 presents a conclusion to this work.

6.2 Motivating Case Study

The Dynamic E/T Job Scheduling Problem is motivated by business processes observed in real world. Typically, E/T concept is predominantly used in supply chain management to produce goods just-in-time to reduce inventory (holding) cost. However, here we present a business process as a case study that E/T scheduling can be applied and form an important aspect of providing quality services. For ease of illustration, simplified version of the actual process is provided here. We shall then use the case study as the basis for our proposed solution.

A airport terminal service provider receives orders from their customers (the various airlines) to supply in-flight food and beverages to the flights departing from the airport. The orders are made in advance to the service provider. Each order is a job that the provider must be able to fulfill before the estimated departure time (ETD) of the flights, i.e., the due date of each order. Each job consists of a series of operations (in sequence or in parallel) in order to complete the job. A simplified business process of the catering job is provided in Figure 6.1. In a job, the operations include preparation of ingredients, hot dishes, special meals (e.g. vegetarian, diabetic meals), beverages, tray assembly and finally loading into truck or moved to staging area. Each operation within the job requires use of one or more types of limited resources and may also require more than one unit of each type of resource. For example, preparation of hot dishes may require multiple cooks and stoves.

Based on all the orders received, the provider plans and schedules the jobs in advance.

However, such a process is subjected to highly dynamic business environment as the orders can be amended during the job execution due to changes in business demand. The preparation of food begins three days in advance based on initial planned order. However, the airline may amend the order as it approaches the ETD due to changes in passenger load (paxload). Paxload is highly dynamic in the last few days and even hours due to last minute bookings, cancellations and check-in information. The paxload information may arrive at the provider as late as 40 minutes before the ETD! Such amendment of orders affects the duration of the operations in the job and this is the source of uncertainty that we will deal with in this work.

To further complicate matters, these jobs should complete as close to the due-dates as possible. Early completion of jobs imposes risk to freshness of the products and also occupies staging area and limited resources such as tray tower. Late completion of job is highly detrimental as it may cause flight delays and hence risking the loss of customer's goodwill. The challenge for the provider, not only is to be able to complete the jobs just-in-time but also have an executable strategy for handling jobs in this dynamic environment. The above discussion paves a need for our study of E/T job scheduling under duration uncertainty.

6.3 Problem Definition

We formalize the Dynamic E/T Job Scheduling problem as follows: A project consists of n independent jobs j_1,...,j_n, where each job j represents an instance (or case) of the business process with a due date D_j (jobs have distinct due dates), a set of operations O_{jh}, where h_j denotes the hth operation of job j; and a precedence constraint PR matrix of size $O_{jh} \times O_{jh}$ to provide flexibility to allow parallel operations in the job. Each operation O_{jh} has processing time $p_{jh} \epsilon \mathbb{R}^+$ and has resource requirements RQ_{jhk} for resource type R_k. The processing time is subject to uncertainty by a random variable $\tilde{z}$, i.e. $\tilde{p_{jh}} = p^0_{jh} + \tilde{z}$. p^0_{jh} denotes the deterministic duration.

We are given a set of k type of resources, which any of its members of type k are capable of executing the same task. Each type of resource is denoted by R_k. Each resource R_k has capacity RC_k. The resources are subjected to constraint $\sum_{jh \epsilon a_t} RQ_{jhk} \leq RC_k \; \forall t, k$, where resource usage for each resource type k does not exceed the capacity at any point of time during the execution

of the set of activities in which a_t is the set of active activities at time t.

We define cost of earliness or tardiness, $T_j \epsilon \mathbb{R}^+$ to measure the penalty that a job can incur if it is completed before or after it's due date D_j. We obtain T_j from two functions, an earliness function W_j and lateness function Y_j. This model of using W_j and Y_j functions supports both common and distinct cost functions problems. We assume that both functions can be estimated or obtained from historical data. Let $\tilde{C_j}$ be the variable completion time of job j due to uncertainties.

1. If job j completes before its due date, then $\tilde{T_j} = W_j(D_j - \tilde{C_j})$
2. If job j completes on or after its due date, then $\tilde{T_j} = Y_j(\tilde{C_j} - D_j)$.

Since each project contains j jobs, the E/T cost of a project, $\tilde{T_j}$ is also the sum of the E/T costs of all jobs.

$$\tilde{T} = \sum_{j \in J} \tilde{T_j} \tag{6.1}$$

The goal is to find an execution policy for the project with a given risk level $\epsilon \in [0, 1]$, the policy yields the minimum robust E/T cost T^*_{min}, such that there is a probability of $(1 - \epsilon)$ that the actual cost is less than or equals to T^*.

$$T^* = E[\tilde{T}] + \sqrt{1 - \epsilon/\epsilon}\sqrt{Var[\tilde{T}]} \tag{6.2}$$

We highlight the uniqueness of our problem with respect to various standard problems in the literature. With respect to Resource-Constrained Project Scheduling Problem (RCPSP), our problem includes an additional structure of a job which links a series of activities (or operations) together and has a due date. Our problem also differs from a standard Job-Shop Scheduling (JSP) problem as the jobs has due dates, and each operation within the jobs may require one or more units of k types of resources. The standard JSP, on the other hand, involves only single unit of a single resource in each operation. In addition, our work differs from standard E/T scheduling (comprehensive survey found in [3]) as we takes into considerations, duration uncertainty in dynamic environment as compared to deterministic problems. Although scheduling under uncertainty has been studied in RCPSP and JSP (in [21, 22, 38] and

[5] respectively), these works are concerned with completing project in shortest possible time and do not have earliness considerations that is important to service timeliness.

6.4 Proposed Approach to E/T Case Study

In this section, we show our proposed approach in solving the Dynamic E/T Job Scheduling problem for the case-study described in section 6.2. First, some assumptions/characteristics are given as follows.

- The mean and variance of the processing time uncertainty of each operation O_{jh} takes normal distribution $N(0, \sigma_{jh})$.
- The earliness and tardiness cost functions are known or can be derived from historical data. Both functions are step functions with respect to discrete time of completion of a job. We model the cost functions with step functions because penalty costs such as holding cost of the goods are usually incurred hourly or daily, and not continuously in time. In extreme case where the cost of earliness and tardiness is continuous, we are still able to apply our method by using discrete approximation.
- All activities are non-preemptive. Resources assigned to a specific operation work for the entire operation duration.
- Late completion of jobs has more severe penalty compared to early completion of jobs, i.e. function Y has a steeper slope than function W. This assumption reflect practical situation where loss of customer satisfaction is more detrimental than holding cost. We use this as a guide to our search algorithm and also minimize the amount of tardiness.
- Operations in all jobs are of equal priority. Search algorithm may decide to place the operations in any sequence as long precedence constraints between operations are preserved.

6.4.1 Overview

Figure 6.2 shows an overview of our approach to solve the Dynamic E/T Job Scheduling problem. We propose a local search algorithm that searches for a POS that gives minimum robust E/T cost, T^*_{min}.

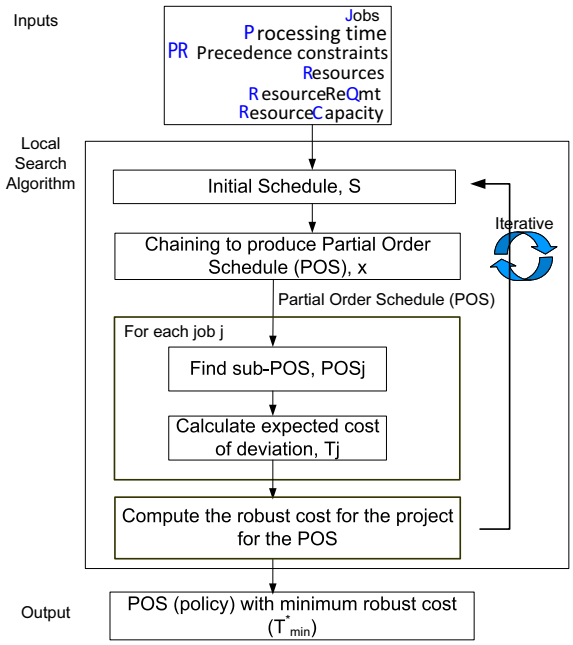


Figure 6.2: Overview of the approach to E/T job scheduling problem

Algorithm 7 shows the detailed algorithm for our local search function. We first randomly form an initial activity list(AL) and obtain an initial schedule S (Line 01-02). Schedule S can be obtained either by a standard scheduling heuristic such as serial schedule generation scheme (no idle time inserted) or any E/T scheduling methods with inserted idle time [29]. With schedule S, forward-backward-chaining (FBC) is applied to obtain a POS for the schedule (Line 03). Line 05-09 shows how we obtain the expected E/T cost $E(\tilde{T})$ for the entire project. For each job j, we find the sub-POS involving the activities in the jobs (details in section 6.4.2). The use of sub-POS is required to compute the expected cost of E/T for each job since the the completion time of the job also depends on the consumption of resources by activities in other jobs. The cost of E/T for job j is computed using earliness and lateness functions (W_j and Y_j respectively) (Line 07) and the expected E/T cost of the project is the sum of $E(\tilde{T}_j)$ for all jobs (Line 08). We then calculate the robust cost for the project T^*_{now} based on equation 6.2 (Line

10).

Line 11-21 is the local search algorithm in which we perform the search over *MaxIteration*. Line 12 to 15 show that we replace POS_{min} and T^*_{min} whenever we find one that gives a smaller robust cost of E/T. Lines $16-17$ shows the local search moves where two operations are randomly selected for swap to produce a new activity list AL'. We then repeat the search process using the new activity list AL' (Lines $18-21$). At the end of $MaxIteration$, we select the POS (execution policy) that provides the minimum robust E/T cost for the project.

Algorithm 7 Algorithm to find POS with minimum robust E/T cost

1: Randomly form an initial activity list AL
2: Find a schedule S randomly according to AL
3: POS $\leftarrow$ Chaining(S)
4: $T \leftarrow 0$
5: **for** each job j **do**
6: $\quad POS_j \leftarrow FindSubPOS(POS, j)$
7: $\quad T_j \leftarrow$ ComputeExpectedETCost(POS_j, α_j, β_j)
8: $\quad T \leftarrow T + T_j$
9: **end for**
10: Compute robust cost T^*_{now}
11: **for** $i = 1$ to MxIteration **do**
12: $\quad$ **if** $T^*_{now} \leq T^*_{min}$ **then**
13: $\quad\quad T^*_{min} \leftarrow T^*_{now}$
14: $\quad\quad POS_{min} \leftarrow POS'$
15: $\quad$ **end if**
16: $\quad$ Randomly pick two operations a and b from AL
17: $\quad$ Swap a and b in AL as AL'
18: $\quad$ Find a schedule S' according to AL'
19: $\quad POS' \leftarrow FBC(S')$
20: $\quad$ Find T^*_{now}
21: **end for**
22: Output a POS_{min} with T^*_{min}

6.4.2 Finding sub-POS

Our approach relies on use of sub-POS to find the expected cost of E/T of each job. The sub-POS for job j (i.e., POS_j) consists of all the operations in the job as well as the other preceding operations from other jobs in the main POS that job j depends on due to sharing of the common pool of resources. Illustrating with an example, figure 6.3 shows an example of

Algorithm 8 Algorithm to find sub-POS for job *j*

```
1: Given a POS and job j
2: Find the activity a that represents the last operation of job j
3: Add a to POS_j
4: Begin Function addPredecessorOf(a)
5: Vector v ← get predecessors of a
6: if v is not empty then
7:    for all elements e in v do
8:       add e to POS_j
9:       chain e to a
10:      addPredecessorOf(e)
11:   end for
12: else
13:   exit function addParents
14: end if
15: endFunction
16: Return POS_j
```

three jobs and its operations. Suppose the operations are translated to activities numbered 1 to 8 and has a possible POS represented in Figure 6.4. The sub-POS for jobs j_1, j_2 and j_3 consist of activities (1,4,7,2,8,3), (1,4,7,2,8,5,3,6), (1,4,7,2,8) respectively. This is obtained by using the last operation of each job and working backwards recursively to include any preceding activity (parent) in the POS to be included in the sub-POS. An algorithm for determining the sub-POS is shown in Algorithm 8.

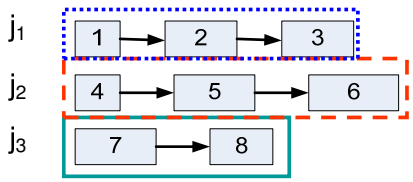


Figure 6.3: Example of three jobs

The use of sub-POS provides greater visibility to scheduling as one gain visibility to the dependencies on other operations (of another job). This is particularly important in applications whereby one would like to anticipate and be able to update airlines (i.e., customers) the status of the order and also be able to plan for the next chain of activities, e.g. dispatching service.

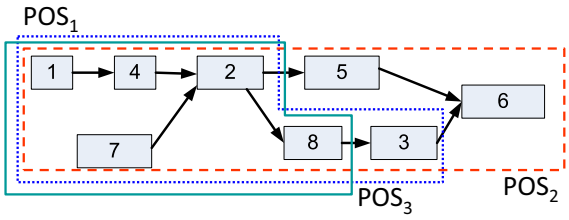


Figure 6.4: Example of sub-POS for three jobs in Figure 6.3

6.4.3 Expected Cost of E/T for Each Job

The cost of E/T of a job depends on whether job completes before or after the job due date, D_j and in the dynamic environment, the job completion time $\tilde{C}_j$ is variable. We first consider a simple model where the cost functions are linear functions of the differences between job due date and completion time.

In such a situation, we have two mutually exclusive cases which either $\tilde{C}_j$ lies before or after due date. We could model the expected cost of E/T each job, $E(\tilde{T}_j)$ by

$$E(\tilde{T}_j) = P(\tilde{C}_j \leq D_j) * \alpha(D_j - \tilde{C}_j) + (1 - P(\tilde{C}_j \leq D_j)) * \beta(\tilde{C}_j - D_j) \tag{6.3}$$

Similarly, this probabilistic model is also applicable for other continuous cost functions such as quadratic earliness and lateness cost functions:

$$E(\tilde{T}_j) = P(\tilde{C}_j \leq D_j) * \alpha(D_j - \tilde{C}_j)^2 + (1 - P(\tilde{C}_j \leq D_j)) * \beta(\tilde{C}_j - D_j)^2 \tag{6.4}$$

For a special case when cost functions are step functions, then equation can be written as:

$$E(\tilde{T}_j) = P(\tilde{C}_j \leq t_1) * Cost(C_j \leq t_1) + \sum_{i=2}^{5} P(t_{i-1} \leq C_j \leq t_i) * Cost(t_{i-1} \leq C_j \leq t_i) + (1 - P(\tilde{C}_j < t_5)) * Cost(\tilde{C}_j \geq t_5) \tag{6.5}$$

6.4.4 Robust Cost for the Project

To evaluate each execution policy (represented by the POS) for a given risk level, we compute T^*_{min} computed based on Equation 6.2. Using this fitness function, we aim to achieve, as the output of search process, an execution policy that takes uncertainty into considerations and yet provide a minimum robust cost of earliness and tardiness.

6.5 Experimental Results

In this section we will present the experimental results of applying the local search algorithm to solve the E/T scheduling problem with durational uncertainties. We run our search algorithm based on problem instances derived from the benchmark sets for RCPSP/max J10 as specified in the PSPLib. We used the activity durations, resource requirements and resource capacities from the benchmark sets. For each instance, we added the layer of jobs, added due dates for each jobs and changed the time-lag parameters to suit the requirements of our test. For each test instance, we have 4 jobs and each job has either 2 or 3 operations. Each operations requires 1 or more units of k types of resources. 5 different types of resources are used.

For each operations O_{jh}, we set the expected processing time p_{jh} of the stochastic duration as the corresponding deterministic duration given by the benchmarks and assume that duration uncertainty $\tilde{z_{jh}}$ is normally distributed with mean 0. We run the algorithm across 4 increasing levels of risk ϵ = 0.01, 0.5, 0.1, 0.2 and 4 different local search iterations $MaxIteration$ = 50, 100, 250, 500. To reduce the possible effect of random factors during the search process on final results, we average over 5 random executions for each problem instance.

In our tests, we used linear costs functions similar where factors α and β are set to 0.5 and 0.8 respectively. We used simulation technique to obtain the probabilities required for computation of expected cost for each job. We achieved this by providing 100 actual realizations of the instance for each POS generated by the algorithm. The expected cost of the project $E(\tilde{T})$ is then obtained from summation of expected cost of the jobs and the variance of the E/T cost of the project $Var(\tilde{T})$ is derived through the simulation. The robust E/T cost T^* for each POS is then calculated based on 6.2.

Firstly, we present the results of sensitivity test on the effect of the risk level ϵ on the robust cost. Figure 6.5 shows 5 selected instances from our test results with 250 local search iterations. We observe that the robust cost decreases as the level of risk increases. This is important to the planner as a decision maker on which strategy to adopt to improve the service quality in his organization, i.e., an policy that yields lower robust cost by taking a higher risk or vice versa. The experiments also show that this results is consistent with different instances of the problems.

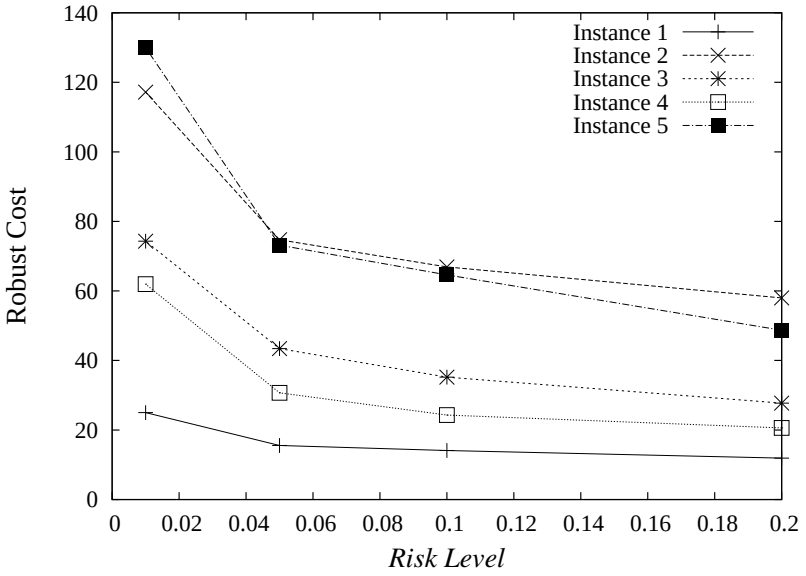


Figure 6.5: Selected instances of robust cost versus level of risk

Next, we evaluate the effectiveness of our algorithm by executing the experiment using different cycles of local search.Figure 6.6 shows 5 selected instances from our test results with $\epsilon = 0.2$. The results shows that with 250 iterations of local search, the algorithm is able to provide a good convergence to minimum robust cost.

In our experiment, we also observe that POS that yields the minimum robust cost is not necessarily the same POS that yields the minimum expected cost of the project. We observe that the POS giving the minimum robust cost may have a higher expected cost as oppose to another policy with the minimum expected cost. We would like to highlight the value of robust considerations. Suppose the risk level is 0.2. In robust scheduling, we are finding a policy that gives the cost of at most T^*_{min} with probability of 0.8. However, if we search for minimum expected cost of the project, if the variance of the project cost is high, we may end up with

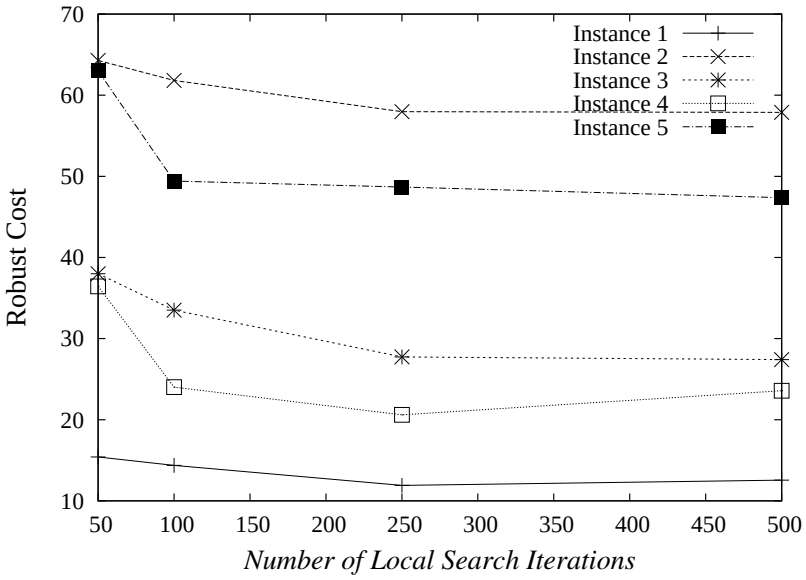


Figure 6.6: Instances of test cases showing the ability of algorithm to converge to a minimum robust cost

higher total cost over all realizations of the project. As such, the policy provided by the robust scheduling provide the hedge against adversity and hence valuable to the decision makers.

6.6 Conclusion

We have shown in this chapter, an approach to evaluate the robust cost of earliness and lateness for a JIT project with duration uncertainty. We discussed that E/T job scheduling problem under uncertainty is an important consideration as it has many possible applications in real-life business processes in improving service delivery. We proposed a local search algorithm in search for a schedule with minimum robust E/T cost for all jobs in the project. By using the concept of partial order schedule (POS), we also provide an executable strategy as a guidance to react to uncertainty. In addition, we show that we could add visibility to service provider by having the ability to compute the job completion time for each job.

Going forward, this work provides many potential for future work. It would be realistic to explore the possibilities of grouping of jobs that are similar in nature. In the example of flight catering business, to allow the operations to fulfill the orders that require similar products to be processed together. The former has interesting real-life application that imposes a maximum

duration between start and expected end time of a job. The latter is applicable to uncertainty such as flight delay (hence delay of the catering job is required). We could also extend our work to include more studies on how to improve the search algorithm by applying heuristics such as tabu-search.

Chapter 7

Conclusion and Future Research

To the best of our knowledge, this thesis is the first work dealing with stochastic durations, resource breakdowns, and minimum/maximum time lags simultaneously in project scheduling. In this final chapter, I will conclude by describing the progress made in terms of development of various algorithms and its application to in service industry domain. I will also suggest some future research directions that could provide the next steps along the path.

7.1 Conclusion

The aim of this thesis has been to propose methods for scheduling RCPSP/max problems under uncertainty. Given a level of risk $0 < \varepsilon \leq 1$ chosen by the planner, data uncertainties modeled by stochastic activity durations and uncertain resource availabilities, we investigated the problem of finding the minimum robust makespan for RCPSP/max and extended local search methods to find the robust schedule strategy (POS) such that when uncertainty is dynamically realized, the execution strategy will result in a solution whose value is as good as the robust makespan. We first put forward General Non-linear Approximation as a decision rule utilized in scheduling to help specify the start times for all activities with respect to execution policy and dynamic realizations of data uncertainty. Based on the decision rule, fitness function can be derived to evaluate robustness, which was finally integrated into a local search method to produce the solution with robust makespan [20, 22]. We then applied our model to the MRO

industry context and provide the planner guidance of execution strategies. To count for the effect of resource breakdown, we proposed different chaining procedures to obtain execution strategies based on both resource breakdown and repair distributions [21]. Experimental results illustrate our proposed methods can generate robust execution strategies efficiently and POS can can absorb both resource and durational uncertainties. With enhancement techniques, local search can produce provider tighter bounds on robust makespan and better POSs. In addition, we applied our model in the service industry context and evaluated the robust cost of real-world business processes [54].

7.2 Future Research

Following the investigations described in this thesis, we summarize the avenues of future research as follows.

- **Considering Project Network Disruptions:** In real scheduling environment, there are possible scenarios that activities may be added or removed, or temporal constraints may be revised. However, this thesis focuses on a deterministic structure of the project network. Due to the practical importance, it is therefore desirable to extend our methodologies to modeling and solving project scheduling problems with activity disruptions, resource disruptions and project network disruptions at the same time.

- **Integrating Reactive mechanisms with Robust scheduling approaches:** In this thesis, we solve project scheduling problem from the perspective of proactive scheduling and obtain robust execution strategy offline. Our future research efforts aim at intergrading robust local search models with reactive scheduling mechanisms, implementing the obtained execution strategy and evaluating its online behavior.

Appendix A

System Implementation Diagram

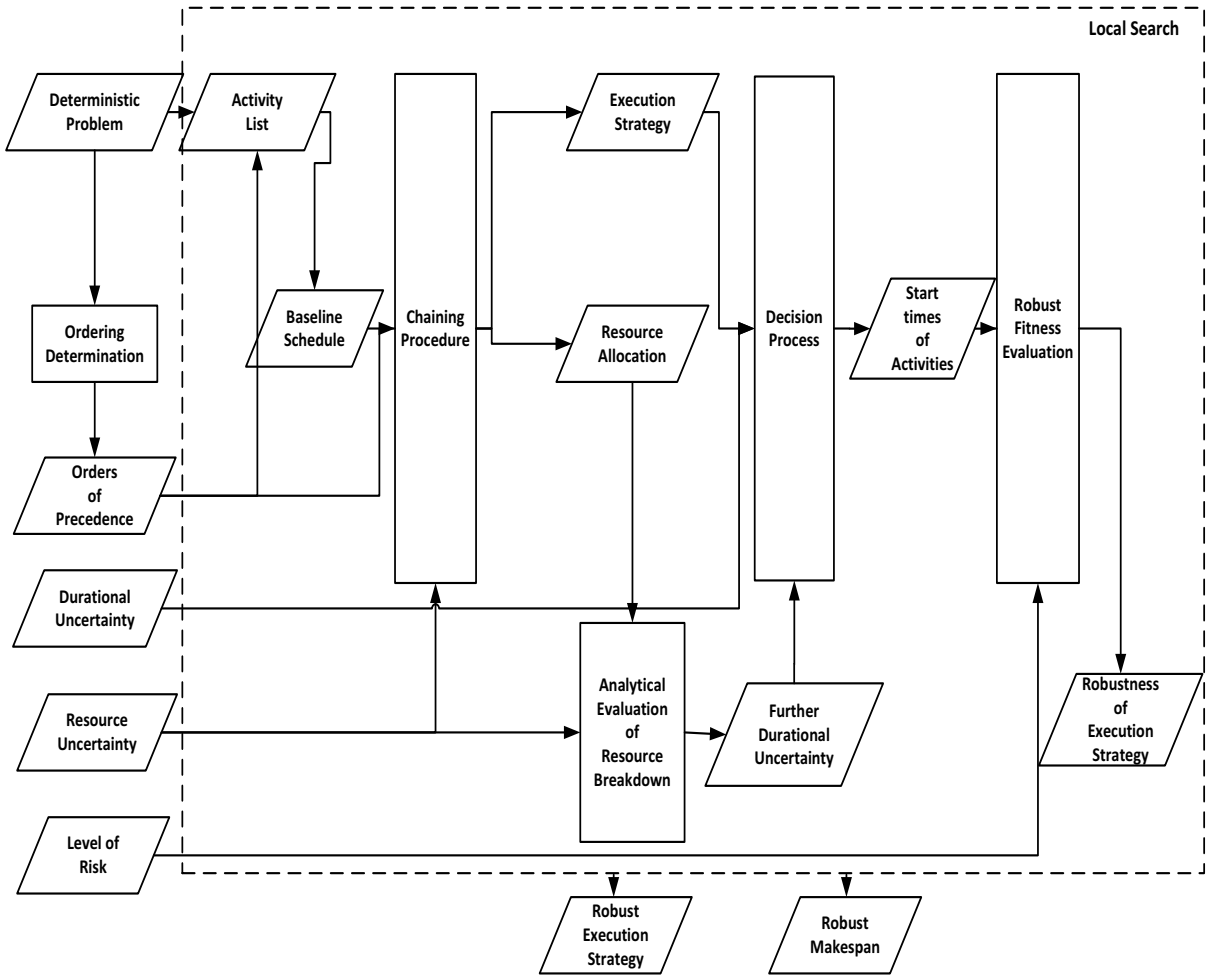


Figure A.1: System Implementation.

A.1 Inputs/Outputs of the System

- Inputs of the System:
  - Deterministic RCPSP/max Problem Instance
  - Mean and Variance Values of Durational Uncertainty for Each Activity
  - Exponential Distributions for Time between Failures and Repairing Time (once Failed) of Each Resource
- Outputs of the System:
  - Robust Makespan
  - Robust Execution Strategy that can Achieves Robust Makespan

A.2 Guidelines on Implementing the System

The following provides guidelines on implementing the system in Figure A.1.

1. **Deterministic Problem**
   This is the Resource-Constrained Project Scheduling Problem with minimum and maximum time lags, or RCPSP/max, in a deterministic setting without uncertainty.
2. **Durational Uncertainty**
   Durations of activities are modeled as stochastic variables with known values of means and variances for the uncertain part.
3. **Resource Uncertainty**
   The time between breakdowns and the time needed to be repaired (once it breaks down) for each resource follow exponentially distributions.
4. **Level of Risk**
   A risk value between 0 and 1 prescribed by the planner.

5. **Execution Strategy**

   Partial Order Schedule, or POS.

6. **Robust Makespan**

   The minimum fitness value of all POSs generated by local search.

7. **Robust Execution Strategy**

   The POS with the minimum fitness value.

8. **Activity List**

   An activity list is a sequence of activities that satisfy the non-negative minimal time lag constraint defined in the deterministic problem. The system performs local search on activity lists. Optionally, the information on orders of activity (output of the process of Ordering Determination) can also be used to focus local search on activity lists.

9. **Baseline Schedule**

   A baseline schedule is an assignment of start times to all activities that satisfy temporal constraints and resource constraints defined in the deterministic problem. A baseline schedule can be produced based on an activity list.

10. **Ordering Determination**

    This is an optional process that exploits the precedence ordering between activities. The output of the process is activity pairs with precedence ordering.

11. **Chaining Procedure**

    The inputs of the chaining procedure are:

    - A baseline schedule generated from the deterministic problem;
    - Orders of pairs of activities generated from the process of Ordering Determination (optional);
    - Breakdown parameters under resource uncertainty.

    The outputs of the process are:

    - POS constructed from that baseline schedule;

- Resource allocations for each activity.

Three chaining procedures are proposed in this work: Robustness-Feedback Resource Chaining (see Algorithm 2), Forward-Backward-Chaining (see Algorithms 3 and 4) and Resource-Breakdown-Aware Chaining (see Algorithm 5).

12. **Decision Process**

The inputs of the decision process are:

- Execution strategy;
- Durational uncertainty;
- Further durational uncertainty due to resource breakdowns.

The outputs of the process are:

- Representation of start times for each activity;
- Representation of makespan which is a random variable in a stochastic scheduling environment.

Decision process computes the start times of activities with respect to execution startegy and dynamic realizations of data uncertainty. Two decision rules can be applied: Segregated Linear Approximation (SLA) in Section 3.4.2 and General Non Linear Approximation(GNLA) in Section 4.2.1.

13. **Analytical Evaluation of Resource Breakdown**

The inputs of the process of analytical evaluation of resource breakdown are:

- Exponential distributions for the time between failures and the repairing time of each resource;
- Resource assignment for each activity.

The outputs of the process are:

- Mean and variance values of durational perturbations for each activity due to resource breakdowns and repairs during the execution of that activity.

14. **Robust Fitness Evaluation**

The inputs of process of robust fitness evaluation are:

- Representation of makespan (as a random variable) of POS, derived from the procedure of decision process;
- Robust fitness function (see Eqn 5.21);
- A level of risk between 0 and 1 prescribed by the planner.

The output of the process is:

- Fitness of that POS, a value that any schedule instantiated from that POS can finish before the fitness value within probability guarantee.

Bibliography

[1] Shoukat Ali, Howard Jay Siegel, and Anthony A. Maciejewski. The robustness of resource allocation in parallel and distributed computing systems. *Parallel and Distributed Computing, International Symposium on*, 0:2–10, 2004.

[2] Haldun Aytug, Mark A. Lawley, Kenneth McKay, Shantha Mohan, and Reha Uzsoy. Executing production schedules in the face of uncertainties: A review and some future directions. In *European Journal of Operational Research*, volume 165(1), pages 86–110, Feb 2005.

[3] Kenneth R. Baker and Gary D. Scudder. Sequencing with earliness and tardiness penalties: A review. *Operation Research*, 38(1):22–36, Jan-Feb 1990.

[4] M. Bartusch, R. H. Mohring, and F. J. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16(1-4):201–240, 1988.

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

[6] A. Ben-Tal and A. Nemirovski. Robust optimization - methodology and applications. *Mathematical Programming*, 92:453–480, 2002.

[7] Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98:49–71, 2003.

[8] Julien Bidot, Thierry Vidal, Philippe Laborie, and J. Christopher Beck. A theoretic and practical framework for scheduling in a stochastic environment. *Journal of Scheduling*, 12:315–344, June 2009.

[9] Amedeo Cesta, Angelo Oddi, and Stephen F. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8(1):109–136, January 2002.

[10] Xin Chen, Melvyn Sim, Peng Sun, and Jiawei Zhang. A linear decision-based approximation approach to stochastic programming. *Operations Research*, 56(2):344–357, 2008.

[11] Charles E. Clark. The Greatest of a Finite Set of Random Variables. *Operations Research*, 9:145–162, 1961.

[12] R.L. Daniels and J.E. Carrillo. Beta-robust scheduling for single-machine systems with uncertain processing times. *IIE Transactions*, pages 977–985, 1997.

[13] Andrew J. Davenport, Christophe Gefflot, and J. Christopher Beck. Slack-based techniques for robust schedules. In *Proceedings of the 6th European Conferences on Planning (ECP)*, 2001.

[14] Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. In *FOCS*, pages 208–217, 2004.

[15] Filip Deblaere, Erik Demeulemeester, and Willy Herroelen. Proactive policies for the stochastic resource-constrained project scheduling problem. *European Journal of Operational Research*, 214(2):308–316, 2011.

[16] Filip Deblaere, Erik Demeulemeester, Willy Herroelen, and Stijn V. Vonder. Robust Resource Allocation Decisions in Resource-Constrained Projects. *Robust Resource Allocation Decisions in Resource-Constrained Projects*, 38(1):5–37, April 2007.

[17] Erik L. Demeulemeester and Willy S. Herroelen. *Project scheduling : a research handbook*. Kluwer Academic Publishers, Boston, 2002.

[18] Ulrich Dorndorf, Erwin Pesch, and Toàn Phan-Huy. A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. *Manage. Sci.*, 46(10):1365–1384, October 2000.

[19] C. Eden, T. Williams, F. Ackermann, and S. Howick. The role of feedback dynamics in disruption and delay on the nature of disruption and delay (d&d) in major projects. *Journal of the Operational Research Society*, 51(3):291–300, March 2000. Theoretical Paper.

[20] Na Fu, Hoong Chuin Lau, Pradeep Varakantham, and Fei Xiao. Robust local search for solving rcpsp/max with durational uncertainty. *Journal of Artificial Intelligence Research (JAIR)*, 43:43–86, 2012.

[21] Na Fu, Hoong Chuin Lau, and Fei Xiao. Generating robust schedules subject to resource and duration uncertainties. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*, pages 83–90, 2008.

[22] Na Fu, Pradeep Varakantham, and Hoong Chuin Lau. Towards finding robust execution strategies for rcpsp/max with durational uncertainty. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*, pages 73–80, 2010.

[23] Eli M. Goldratt. *Critical Chain*. The North River Press, Great Barrington, 1997.

[24] Jane N. Hagstrom. Computational complexity of pert problems. *Networks*, 18:139–147, 1988.

[25] Willy Herroelen and Roel Leus. On the merits and pitfalls of critical chain scheduling. *Journal of Operations Management*, 19(5):559 – 577, 2001.

[26] Willy Herroelen and Roel Leus. Project scheduling under uncertainty: Survey and research potentials. In *European Journal of Operational Research*, volume 165(2), pages 289–306, Sept 2005.

[27] Frederick S. Hillier, Gerald J. Lieberman, Frederick Hillier, and Gerald Lieberman. *MP Introduction to Operations Research*. McGraw-Hill Science/Engineering/Math, July 2004.

[28] SL Janak, XX Lin, and CA Floudas. A new robust optimization approach for scheduling under uncertainty :ii. uncertainty with known probability distribution. *Computers and Chemical Engineering*, 31:171–195, 2007.

[29] John J. Kanet and V. Sriharan. Scheduling with inserted idle time: Problem taxonomy and literature review. *Operations Research*, 48(1):099–110, 2000.

[30] R. Kolisch and S. Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 2005.

[31] R. Kolisch, C. Schwindt, and A. Sprecher. *Benchmark Instances for Project Scheduling Problems*, pages 197–212. Kluwer Academic Publishers, Boston, 1998.

[32] Rainer Kolisch and Sonke Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174(1):23–37, October 2006.

[33] P. Kouvelis, R. L. Daniels, and G. Vairaktarakis. Robust scheduling of a two-machine flow shop with uncertain processing times. *IIE Transactions*, 32:421–432, 2000.

[34] Olivier Lambrechts, Erik Demeulemeester, and Willy Herroelen. Exact and suboptimal reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. Technical report, 2007.

[35] Olivier Lambrechts, Erik Demeulemeester, and Willy Herroelen. Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. *Journal of Scheduling*, 11:121–136, 2008. 10.1007/s10951-007-0021-0.

[36] Olivier Lambrechts, Erik Demeulemeester, and Willy Herroelen. A tabu search procedure for developing robust predictive project schedules. *International Journal of Production Economics*, 111(2):493–508, February 2008.

[37] Olivier Lambrechts, Erik Demeulemeester, and Willy Herroelen. Time slack-based techniques for robust project scheduling subject to resource uncertainty. *Annals of Operations Research*, 186:443–464, 2011.

[38] Hoong Chuin Lau, Thomas Ou, and Fei Xiao. Robust local search and its application to generating robust schedules. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*, pages 208–215, 2007.

[39] Haim Levy. *Stochastic Dominance: Investment Decision Making under Uncertainty (Studies in Risk and Uncertainty)*. Springer, 2nd edition, February 2006.

[40] Zukui Li and Marianthi G. Ierapetritou. Robust optimization for process scheduling under uncertainty. *Industrial and Engineering Chemistry Research*, 47(12):4148–4157, 2008.

[41] Michele Lombardi and Michela Milano. A precedence constraint posting approach for the rcpsp with time lags and variable durations. In *Proceedings of the 15th international conference on Principles and practice of constraint programming*, CP'09, pages 569–583, Berlin, Heidelberg, 2009. Springer-Verlag.

[42] Rolf H. Möhring. Scheduling under uncertainty: Bounding the makespan distribution. In *Computational Discrete Mathematics*, pages 79–97, 2001.

[43] Rolf H. Mohring and Frederik Stork. Linear preselective policies for stochastic project scheduling. *Mathematical Methods of Operations Research*, 52(3):501–515, 2000.

[44] Paul Morris and Nicola Muscettola. Temporal dynamic controllability revisited. In *Proceedings of the 20th National Conference on Artificial Intelligence*, pages 1193–1198. AAAI Press, 2005.

[45] J. M. Mulvey, R. J. Vanderbei, and S. J. Zenios. Robust optimization of large-scale systems. *Operations Research*, 43, 1995.

[46] K. Neumann, C. Schwindt, and J. Zimmermann. Resource-constrained project scheduling with time windows. *International Series in Operations Research and Management Science*, 92:375–408, 2006.

[47] Nicola Policella, Amedeo Cesta, Angelo Oddi, and Stephen Smith. Solve-and-robustify. *Journal of Scheduling*, 12:299–314, 2009. 10.1007/s10951-008-0091-7.

[48] Nicola Policella, Amedeo Cesta, Angelo Oddi, and Stephen F. Smith. From precedence constraint posting to partial order schedules: A csp approach to robust scheduling. *AI Communications*, 20:163–180, August 2007.

[49] Nicola Policella, Stephen F. Smith, Amedeo Cesta, and Angelo Oddi. Generating robust schedules through temporal flexibility. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*, pages 209–218, 2004.

[50] Riccardo Rasconi, Amedeo Cesta, and Nicola Policella. Validating scheduling approaches against executional uncertainty. *Journal of Intelligent Manufacturing*, 21(1):49–64, 2010.

[51] John A. Rice. *Mathematical Statistics and Data Analysis*. Duxbury Press, April 2001.

[52] Inés González Rodríguez, Camino R. Vela, Jorge Puente, and Alejandro Hernández-Arauzo. Improved local search for job shop scheduling with uncertain durations. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.

[53] Tristan B. Smith. An effective algorithm for project scheduling with arbitrary temporal constraints. In *In: Proceedings of the 19 th National Conference on Artificial Intelligence. (2004*, pages 544–549, 2004.

[54] Kar Way Tan, Na Fu, and Hoong Chuin Lau. Improving service through just-in-time concept in a dynamic operational environment. In *Hawaii International Conference on System Sciences (HICSS)*, pages 1–9, 2011.

[55] Stijn Van de Vonder, Erik Demeulemeester, and Willy Herroelen. Proactive heuristic procedures for robust project scheduling: An experimental analysis. *European Journal of Operational Research*, 189(3):723–733, 2008.

[56] Thierry Vidal and Helene Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 11:23–45, 1999.

[57] Stijn Vonder, Erik Demeulemeester, and Willy Herroelen. A classification of predictive-reactive project scheduling procedures. *Journal of Scheduling*, 10(3):195–207, 2007.

[58] Christos Voudouris, G. Owusu, R. Dorne, and D. Lesaint. *Service Chain Management*. Springer-Verlag, Berlin, 2008.

[59] Benjamin W. Wah and Dong Xin. Optimization of bounds in temporal flexible planning with dynamic controllability. *IEEE International Conference on Tools with Artificial Intelligence*, 0:40–48, 2004.

[60] Jean.-Paul. Watson, Laura Barbulescu, L. Darrell Whitley, and Adele E. Howe. Contrasting structured and random permutation flow-shop scheduling problems: Search-space topology and algorithm performance. *INFORMS Journal on Computing*, 14:98–123, 2002.

[61] Christine Wei Wu, Kenneth N. Brown, and J. Christopher Beck. Scheduling with uncertain durations: Modeling beta-robust scheduling with constraints. *Computers and Operations Research*, 36:2348–2356, 2009.

[62] G Zhu, Jf Bard, and G Yu. Disruption management for resource-constrained project scheduling. *Journal of the Operational Research Society*, 56:365–381, 2005.