# Stochastic Scheduling of Train Maintenance Projects

*Master's Thesis*

Kiriakos Simon Mountakis

# Stochastic Scheduling of Train Maintenance Projects

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Kiriakos Simon Mountakis
born in Chania, Hellas

Algorithmics Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

NedTrain
Postbus 2167
3500 GD Utrecht, the Netherlands
www.nedtrain.nl

Cover picture: NedTrain maintenance depots.

# Stochastic Scheduling of Train Maintenance Projects

Author: Kiriakos Simon Mountakis
Student id: 4051343
Email: k.s.mountakis@gmail.com

**Abstract**

We study the application of stochastic scheduling methods for dealing with the negative impact of uncertainty on the timely delivery of NedTrain maintenance projects. A stochastic scheduling problem includes a quantification of of uncertainty by representing the duration of a maintenance activity with a probability distribution. The solution is no longer a schedule but a *scheduling policy*: a dynamic process for determining when to execute activities on-the-fly. Our objective is to find a policy that minimizes stochastic tardiness by expectation and also by variance, to minimize risk. We concentrate on the class of Early-Start (ES) policies. An ES policy is a PERT network representation of the projects in which the width of the partial order defined by the precedence edges is restricted such that resource requirement will not exceed resource availability. ES policies enable the use of standard PERT management practices (developed for handling uncertainty in projects with unlimited resources) to projects with limited resources (e.g., NedTrain projects). Simulation is considered as the only practical means to evaluate the objective function in stochastic scheduling. This approach is computationally expensive, especially for large (e.g., NedTrain-specific) instances with activity durations that exhibit high variability. We observe that circuit timing graphs in modern VLSI design use a PERT network-like representational mechanism. Statistical Static Timing Analysis (SSTA) is a new research area with a wealth of methods for solving the PERT problem on massive networks, offering high accuracy at low computational cost. We propose the use of SSTA techniques, treating a given ES policy as a circuit timing graph. This enables us to evaluate the objective function and more importantly to implement informed heuristics with efficiency. As a proof-of-concept we studied the use of Linear-Gaussian SSTA, applicable when activity durations exhibit Gaussian variability and may have linear interdependencies. Our other contribution is to propose efficient enumeration schemes for ES policies, overcoming the infeasibility of the existing scheme on problems of practical size. Experiments suggest that the proposed enumeration schemes in combination with SSTA establish a framework for the design of competitive heuristic solvers, suitable for large-scale problem instances with high durational variability.

Thesis Committee:

| | |
|---|---|
| Chair: | Prof. dr. C. Witteveen, Faculty EEMCS, TU Delft |
| University supervisor: | Dr. T. B. Klos, Faculty EEMCS, TU Delft |
| Company supervisor: | Ir. B. Huisman, NedTrain maintenance development |
| Committee Member: | Prof. dr. ir. K.I. Aardal, Faculty EEMCS, TU Delft |

# Preface

As a graduate student in TU Delft, I was honored with the opportunity to participate in the Algorithmics research group. This thesis is my contribution to a research project carried out in cooperation with NedTrain–a Dutch company that operates the maintenance of the Nederlandse Spoorwegen (NS) train fleet. Michel Wilson proposed the idea of capturing uncertainty in maintenance projects by representing project activities with stochastic variables. This was the point of departure for my thesis which eventually led me to the fascinating field of stochastic project scheduling.

I would like to thank my supervision team, for giving me guidance and keeping me motivated. In particular, I would like to thank Cees Witteveen for his guidance, for accepting me in the Algorithmics research group and for introducing me to NedTrain, Tomas Klos for his useful advice and detailed feedback which helped shape my writing style, Bob Huisman, for the insightful discussions and the pleasant working environment at the NedTrain headquarters in Utrecht, and finally, Michel Wilson for keeping a close eye on my progress, and for our discussions and exchange of ideas.

I would also like to thank my family and friends for their support during my years in TU Delft.

Kiriakos Simon Mountakis
Delft, the Netherlands
February 20, 2013

# Summary

This thesis is part of a research project on the scheduling of maintenance projects at NedTrain facilities. NedTrain is a Dutch company that operates the maintenance of the Nederlandse Spoorwegen (NS) train fleet. In the conventional sense, a schedule specifies the start-times of a set of project activities (i.e., a set of non-divisible units of work to be carried out in a non-preemptive fashion) in a way that complies with one or more types of constraints. The concept of preparing a schedule entails an unrealistic assumption: that the exact value of project parameters such as the durations of activities are known in advance. In practice, however, most (if not all) projects execute in an uncertain environment. Some activities usually take longer or shorter than expected, equipment breaks down unexpectedly, and so on. As a result, in the schedule that is actually realized, one or more projects violate their delivery due-date.

In an effort to cope with uncertainty in maintenance projects we study the application of *stochastic scheduling* methods. What is assumed to be known about the duration of an activity is not its exact value, but a probability distribution. It should be noted that a train (maintenance project) must undergo inspection before the set of necessary repairs to be performed becomes known. Therefore, some activities of a maintenance project exhibit severe variability of duration. However, it is assumed that expert judgement in combination with past historical data can enable the stochastic quantification of this variability. The solution to a stochastic scheduling problem is no longer a schedule but a *scheduling policy*: a set of rules according to which the start time of each activity will be determined on-the-fly, based on the input problem specification but also on the observed durations of activities that have completed so far. In this stochastic setting, our objective is to find a *robust* policy: one that minimizes the *stochastic tardiness* by expectation, but also by variance (in order to reduce risk). The implementation of a robust scheduling policy by foremen and their workers will eventually lead to the realization of a schedule in which projects are delivered on time, despite the effects of uncertainty.

Several classes of scheduling policies have been proposed in the stochastic scheduling literature. The main body of our work concentrates on the class of Early-Start (ES) policies. An ES policy is a directed acyclic graph in which nodes represent activities. Since activities have stochastic durations, an ES policy can be treated as a so-called PERT network; the schedule may therefore be realized by completing activities in topological order. However,

no activities will have to compete for the use of a resource while doing so. In contrast to a simple PERT network, the width of the partial order defined by the precedence edges in an ES policy is restricted such that at no point in time will resource requirement accumulate beyond availability.[1] The class of ES policies is attractive from a project management point-of-view because it enables the application of standard PERT-based management practices (developed for handling uncertainty in projects with virtually unlimited resources) to projects with limited resources (as is the case with NedTrain projects).

Each scheduling policy corresponds to a stochastic schedule: stochastic variables that represent the start (and completion) times of activities. An objective function is typically defined in terms of the moments of (some of) these variables. This information can also be used in a heuristic during the construction of a policy. Monte Carlo simulation is generally acknowledged as the only practical means for estimating the required information. This approach can be computationally quite expensive and consume most the running time of solver procedures. This is especially true for large problem instances (e.g., NedTrain-specific instances) and even more so when activity durations exhibit high variability.

To resolve this bottleneck we leverage the PERT network nature of ES policies. To estimate the stochastic schedule for a given ES policy amounts to solving the well-known PERT problem. Due to its difficulty and its applicability in various domains, this problem has attracted a lot of research attention since the early 1960s. We observe that circuit timing graphs in modern Very Large Scale Integration (VLSI) design use (an extension of) the PERT network structure as a representational mechanism. Statistical Static Timing Analysis (SSTA) is an active research field with a wealth of techniques for solving (special cases of) the PERT problem on networks of massive size with high accuracy at low computational cost. We suggest that information in the stochastic schedule can be estimated with the application of SSTA techniques by treating a given ES policy as a circuit timing graph. This enables us to evaluate the objective function and more importantly to implement informed heuristics with efficiency, even for large-scale problem instances with high durational variability. As a proof of concept, we study the application of Linear-Gaussian (LG) SSTA: a method that applies to problem instances with activity durations that exhibit Gaussian variability and may have linear interdependencies.

Our second main contribution regards the enumeration of ES policies. The method proposed in the literature is only practical for enumerating policies in the solution-space of very small problem instances. We propose efficient enumeration schemes by extending certain methods that were developed for deterministic scheduling.

Our experiments show that the proposed enumeration schemes achieve a substantial rate even for large instances with more than 600 activities. Furthermore, the computational cost of LG-SSTA is roughly equal to that of only 15 to 20 rounds of simulation. In conclusion, the proposed enumeration schemes in combination with the use of SSTA methods establishes a framework for the design of competitive heuristic solvers, suitable for large-scale problem instances with high durational variability.

[1] For a visual impression of ES policies refer to the example of Appendix A.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 NedTrain maintenance projects

NedTrain is the locomotive and rolling stock maintenance and repair company of the Dutch Railways company, Nederlandse Spoorwegen (NS). NS is the principal passenger railway operator in the Netherlands and its trains operate over the tracks of the Dutch national rail infrastructure (operated by ProRail). NS serves about 1.1 million passengers on a daily basis with a fleet of about 4,800 scheduled trains and consists of the following subdivisions:

- NS Reizigers (NSR): Operates passenger train services.
- NS Hispeed: Operates (in conjunction with NS Reizigers and foreign partners) high-speed train services to France, Germany, and Switzerland.
- Abelio: The international expansion of NS, for passenger transportation in the United Kingdom, Germany, and the Czech Republic.
- NS Poort: Runs the development and exploitation of train stations.
- NS Commercie: Handles product and customer management.
- NedTrain: Operates periodic train fleet maintenance at various depots over the Netherlands.

NedTrain is responsible for maintaining a high availability rate for NS Reizigers and NS HiSpeed trains that comprise a fleet of about 3000 passenger carriages in total. NedTrain provides the following types of maintenace: (*i*) first-line service, (*ii*) technical maintenance, and (*iii*) refurbishment. First-line service involves a daily cleaning and fixing of small technical problems of each train at least once a day at one of thirty facilities throughout the country. Once every couple of months or after a critical part has reached a certain mileage each train goes to one of four NedTrain depots for technical maintenance. Once or twice in its lifetime, a train might have to be refurbished to meet modern standards. Refurbishment is done in large overhaul projects mainly in Haarlem and Tilburg. This thesis only relates to technical maintenance.

Depending on weather conditions and other factors, the number of trains that unedergo technical maintenance in NedTrain depots throughout the country at a certain moment might peek up to 300; i.e., about 10% of the entire fleet. NedTrain has four depots for technical maintenance in The Netherlands: in Amsterdam, Leidschendam, Onnen and Maastricht, and each location specializes in one ore more train types.

### 1.1.1 Maintenance projects

About two weeks prior to an upcomming week, knowledge about which specific trains will arrive at the depot for maintenance becomes available. Each train is expected to arrive on a respective *release-date* and it must be ready for delivery before a respective *due-date*. Each train constitutes a *maintenance project* and multiple projects will typically execute in parallel at the depot.

Depending on its type and the work that was done in its previous visits at the depot the train will undergo a respective set of maintenance activities. An activity corresponds to some unit of maintenance work that will be carried out non-preemptively (i.e., without interruption) and is estimated to take a certain amount of time. Typically about 20% of a maintenance project is spent on inspections, 10% on planned maintenance, 20% on the planned exchange of main parts and 50% on unplanned work based on inspections.

Moreover, each type of activity requires the use of one or more types of resources. For brevity we will only mention here human resources and repair platforms.

Human resources comprise various types of engineers and mechanics that work three 8-hour shifts (starting at 7 am). This allows the depot to remain operational around the clock. The number of available mechanics and engineers does not remain constant during a 24 hour period. In particular, during the first shift human resources have maximum capacity, which reduces during the second shift, and reduces even more during the night shift.

In addition, a depot offers the following types of repair platforms: (*i*) *putspoor* platforms, (*ii*) *kuilwielbank* platforms, (*iii*) *aardwind* platforms, and (*iv*) *ATB* platforms. A putspoor platform is a lowered floor that allows mechanics to work underneath the train. A kuilwielbank platform for re-profiling the outter metal layer of the train wheels. An aardwind platform is used for replacing bogies and an ATB platform is used for testing train safety systems.

As soon as an activity completes, the resources that it requires become free for use by other activities. Scarce resources limit the number of activities that can execute in parallel at anyone time. We say that the execution of activities is subject to *resource constraints*. In addition, activities cannot execute in an arbitrary order because they are interrelated by pairwise *precedence constraints*. For example, activity “engine casing closure” cannot start if activity “engine inspection” is not yet complete.

Having described maintenance projects in detail, let us now proceed with a discussion of the main topic of this thesis: the scheduling and control of these projects in the presence of uncertainty.

### 1.1.2 Project scheduling and control under uncertainty

Before starting our discussion of the main topic of this thesis, let us make clear that we shall refer to the following information as the *data of a maintenance week*: (*i*) maintenance activities and their organization into projects, (*ii*) resource and precedence constraints, and (*iii*) project release and due-dates. Now, having this data it is possible to produce a *baseline schedule* for the maintenance week. The schedule specifies a start-time and a completion-time for every maintenance activity in a way that complies with both the precedence and the resource constraints. One reason for producing a schedule is that it can be used by foremen and workers to decide what needs to be done and when, in order to complete projects on time, given the precedence and resource constraints at the depot.

Thus, NedTrain is interested in the production of good quality schedules. That is, schedules that respect precedence, resource, and release-date constraints and that minimize the violation of due-dates as much as possible. Currently, NedTrain produce baseline schedules largely by hand. In particular, they use Excel along with other tools such as ProPlan and this is a full-time job at NedTrain. But the current approach is not sufficiently effective and maintenance sometimes does not finish in time. For this reason NedTrain (in cooperation with Dutch universities, including TU Delft), initiated a research project (and this thesis is part of it) which focuses on computerized methods for scheduling maintenance projects. *This is expected to make the scheduling of maintenance projects not only more efficient but also more effective*.

However, most (if not all) real-life projects execute in a dynamic environment subject to considerable uncertainty which might stem from various sources. No matter how good a schedule might look, it was produced based on incomplete information regarding the forthcomming week and/or on a set of assumptions that might turn out not to be accurate, as outlined in the following:

- Necessary repair activities are only known after inspection; i.e., if inspection reveals that some part is broken, a corresponding repair activity must be included. This form of uncertainty is characteristic to maintenance projects and is particularly challenging to cope with.
- An additional basic form of uncertainty present in all real-life projects is attributed to activities being natural processes carried out by people. It is thus practically impossible to have an accurate, a-priori knowledge of their duration.
- Another form of uncertainty is attributed to unexpected resource unavailability. Personel might not be present (e.g., due to sickness), required parts might not be supplied on time, and equipment might break down unexpectedly (e.g., because of external factors such as power outages).
- Yet another form of uncertainty is attributed to delayed (or early) train arrivals at the depot (i.e., delayed or early project release-dates) which might result from unexpected changes in the timetables for passenger services.

The purpose of a scheduling process is, in general, the successful time management of one or more projects (in the presence of constraints). Maintaining a time management

point of view, we shall treat all sources of uncertainty in terms of their impact on activity durations. Stated otherwise, we shall only consider what is sometimes refered to in the scheduling literature as *durational uncertainty*. The activity durations that are actually realized during project execution usually deviate from the durations that were originally assumed in order to prepare a baseline schedule. Therefore, the baseline schedule might have to be repaired accordingly in order to comply with precedence and resource constraints. It is thus quite probable that the actual realized schedule will deviate from the pre-computed baseline schedule. This is important, because the compliance of the actual, realized schedule to due-dates might deviate unpredictably from what was originally intended with the baseline schedule.

Uncertainty is the main reason why sometimes maintenance does not finish on time. Thus, a computerized method that produces baseline schedules alone is not sufficient. The method must also cover the issue of controlling the projects in a dynamic execution environment by handling uncertainty. And since we focus exclusively on uncertainty with regard to activity durations, *handling uncertainty effectively means reacting to (realized) duration deviations (from original estimates)*.

## 1.2 Problem description

In this Section we give a non-technical description of the problem at hand by raising three questions, starting with the one below:

**Q1** How can we formulate the data of a maintenance week as a scheduling problem?

Before we continue with the remaining questions, let us introduce a conceptual framework. Let us recall the purpose of finding a schedule in the first place. A schedule specifies a start-time and a completion-time for every maintenance activity (and project). We argue that this information can be seen from the following two viewpoints. One viewpoint is to treat this information as *offline timing predictions* of when projects and activities will start and complete. This viewpoint is useful both for preparing and for controlling the projects. Project management use these predictions in order to: (*i*) communicate with the company's inbound and outbound supply chain and ensure that the necessary resources will be available when needed, (*ii*) identify critical parts of the projects that will require special attention, and (*iii*) to make commitments to the client (i.e., NS) regarding the delivery date of trains. Another viewpoint is to look at this information as *online scheduling directions* that foremen and workers use while executing the projects in order to know which activities to start next such that projects will complete on time.

Let us first focus on the reliability of provided offline timing predictions. We know that a schedule might get repaired multiple times during project execution. If the given offline timing predictions turn out to be very inaccurate then project management is blind with no control over uncertainty. There is no way to prepare for the week, to assess and manage risks, or to make reliable commitments. The outcome of the maintenance week will only be known after the week is over. Furthermore, if predicted start times for remaining activities change all the time (e.g., because the schedule is repaired frequently) this causes confusion

in the depot and hinders performance (this phenomenon is known in the literature as *shop-floor nervousness*). This leads us to the following question:

**Q2** Having formulated a maintenance week as a scheduling problem, how can we obtain a solution that provides *reliable* offline timing predictions for the start and completion times of activities (and overal projects)?

Beyond offline timing predictions, a solution to the scheduling problem also provides online scheduling directions. With a baseline schedule, a foreman can decide immediately which remaining activities to start next because start times are pre-computed and stated explicitly. However, in practice these decisions cannot be taken immediately. If an activity turns out to last much longer than expected it might be necessary to repair the start times (of remaining activities) and the foreman will have to wait until the next decision can be taken. Repairs of online scheduling directions for adapting to change should not take too long, because the time spent might "pushx" the completion of one or more projects closer to their due-date. Furthermore, repairs should not happen all the time because this introduces confusion in the depot (a phenomenon also known in the literature as *shop-floor nervousness*). This leads us to the following question:

**Q3** Having formulated a maintenance week as a scheduling problem, how can we obtain a solution that provides online scheduling directions that can be updated quickly in order to adapt to change (without compromising effectiveness)?

We now proceed with a look at previous work for solving the problem we described here.

## 1.3 Previous work (deterministic scheduling)

Previous work for solving the problem at hand was done by Evers [24] and Wilson [73, 74]. To formulate the data of a maintenance week they use a multi-project problem model. To model precedence constraints they use the Simple Temporal Network (STN) [20] representation which also allows them to include project release and due-dates as hard temporal constraints. Uncertain activity durations are represented by (single-point) most-likely estimates. Solutions to this problem model only include schedules in which all projects finish within their due-dates.

Evers tries to solve NedTrain-specific instances with Integer Programming but finds this to be impractical (due to the size of such instances) and resorts to a heuristic method known as Precedence-Constraint Posting (PCP) [64, 13, 56]. With this method he manages to find *flexible* solutions in a reasonable amount of time (a solution is flexible when it can be easily repaired to adapt to change). Wilson extends the state-of-the-art in PCP with: (*i*) significant improvements in efficiency which allows for quick schedule repairs, and (*ii*) additional improvements regarding flexibility.

Because of the limitations of previous work to fully solve the problem at hand (more on this later), in this thesis we follow a different approach.

## 1.4 Our approach (stochastic scheduling)

Previous work is based on using and extending problem models and solution methods from the research area of *deterministic scheduling*. But (as we discuss later in detail) deterministic scheduling methods fail to give satisfactory answers to all the questions in terms of which we defined the problem at hand in Section 1.2. For this reason we propose a very different approach in this thesis. Our proposal is to use and extend problem models and solution methods from the research area of *stochastic scheduling*.

We use a stochastic scheduling problem model to encode the data of an upcomming maintenance week in which activity durations are only known as stochastic variables with given probability distributions. Uncertainty is included explicitly in the problem formulation and a solver procedure takes it into account in order to find a good solution. Furthermore, because in the presence of uncertainty baseline schedules are difficult to work with, the solution to a stochastic scheduling problem is not a schedule but a less "rigid" artifact that allows for more flexibility in adapting to change during project execution. The solution of a stochastic scheduling problem is a so-called *scheduling policy*. We adopt the notion of scheduling policies as proposed by Möhring et al [49]. Roughly spoken, a scheduling policy is a set of rules, or what we earlier refered to as online scheduling directions, for taking scheduling decisions at certain decision times $t$ and these decisions are based upon the observed past up to $t$, as well as the a-priori knowledge of the input data of the problem. Solutions (i.e., policies) to a stochastic problem are evaluated according to their *robustness*. A policy is robust when its use during project execution will result in a realized schedule of good quality (in our case, one that minimizes violations of project due-dates). The structure of a policy might be hard to imagine at this point, but it will become clear later on. In fact, there have been many "classes" of policies studied in the literature. Once a robust policy has been found as a solution to the input problem, it will remain the same throughout project execution and does not need "repairs" to adapt to change as schedules do.

Beyond online scheduling directions a policy also "contains" what we refered to as offline timing predictions in Section 1.2; i.e., predictions of what the start and completion times for individual activities and whole projects will be in the realized schedule. In contrast to a baseline schedule which contains these predictions in explicit form (i.e., as exact start and completion time values), a policy provides this information implicitly. Since durations are now stochastic variables, we can simulate the policy multiple times and obtain a sample for each start and completion time that describes it as a stochastic variable. Thus instead of having a (single-point) prediction for, e.g., a project's completion as an exact value (which is highly unlikely to be accurate), we have a probability distribution which is accurate and reliable (assuming that the stochastic modeling of durations is accurate in the first place).

The problem formulations are common to all classes of policies; i.e., different policy classes correspond to different types of solutions to stochastic scheduling problems. Each class of policies defines a respective field of research within stochastic scheduling, which deals with various solution methods for this class and also with the theoretical properties of this class. In this thesis we do not only propose the research direction of applying stochastic scheduling to the problem at hand but we also take a first step in that direction. We fix our focus on the class of Earliest-Start (ES) policies. Roughly spoken an ES policy is a set of

precedence constraints between the activities in the input problem. An ES policy containts all precedence constraints given in the input problem, plus a "translation" of given resource constraints in form of extra precedence constraints. Thus, if one executes activities in an order that respects the constraints in the policy, this will result in a realized schedule that also respects resource constraints (regardless of outcome activity durations). This class is attractive from a project management point of view because an ES policy essentially constitutes a PERT network with the special property that it also incorporates resource constraints. Thus, once a robust policy has been produced, project management can simply use the well-established PERT method to prepare, monitor and complete the projects (without having to worry about the scarceness of resources).

Existing solution methods for finding robust ES policies suffer from computational limitations and their use is impractical for problem instances of practical size (such as NedTrain-specific instances). We enable the synthesis of new efficient solution methods for this class of policies and make it possible to handle large stochastic problems (including NedTrain-specific instances). To do this, we observe that an ES policy can be seen as the timing specifications model for a VLSI circuit. This enables us to apply *Statistical Timing Analysis* (STA) methods to evaluate the robustness of enumerated solutions efficiently. Furthermore, we propose methods for enumerating ES policies in the solution-space of the input problem in an efficient manner.

In the following section we make the objective of this thesis more clear by raising research questions that we want to answer.

## 1.5 Research questions and thesis outline

One objective of this thesis is to propose the extension and use of stochastic scheduling methods for tackling the problem at hand. We do this by raising the following research questions. The main objective of this thesis is to take a first step in the proposed direction and give a (partial) answer to each of the research questions.

**RQ1** How can we formulate the data of a maintenance week as a stochastic scheduling problem?

**RQ2** Which classes of scheduling policies are more appealing to NedTrain from a project management point of view?

**RQ3** How can we synthesize efficient and effective solver procedures for appealing policy classes?

1. How can we enumerate policies in the solution-space in an efficient manner?
2. How can we measure the robustness of enumerated policies in an efficient manner?
3. What heuristics can guide solution enumeration towards policies with high robustness?

**On RQ1 and RQ2.** Research questions 1 and 2 are (partially) answered in Chapter 2–*Deterministic Project Scheduling* and Chapter 3–*Stochastic Project Scheduling*. The purpose of Chapter 2 is to: (*i*) describe a deterministic project scheduling problem for encoding the data of a maintenance week (in which most-likely values are used for activity durations) (*ii*) describe previous work for solving this problem, and (*iii*) evaluate previous work against the problem description of Section 1.2 and discuss limitations.

One purpose of Chapter 3 is to extend the problem model described in Chapter 2 into a stochastic variant (in which durations are instead modeled as stochastic variables with given probability distributions), giving an answer to RQ1. Another purpose is to introduce the research area of stochastic project scheduling. Chapter 3 continues with an overview of basic scheduling policy classes, definitions for a robustness metric and the use of simulation for measuring the robustness of a policy. Finally, we justify how the class of Earliest-Start (ES) policies is attractive from a project management point of view, giving an answer to RQ2.

**On RQ3** In this thesis we only attempt to answer research question 3 for the class of ES policies. An answer to RQ3.1 is given in Chapter 4–*Enumerating ES policies* and an answer to RQ3.2 is given in Chapter 5–*Statistical Timing Analysis*. RQ3.3 is not answered in this thesis.

**On RQ3.1.** The context of Chapter 4 is as follows: we are given an input stochastic scheduling problem instance and we want to enumerate ES policies in its solution-space. The chapter starts by describing the most well-known enumeration scheme, developed by Stork [65] in a complete branch & bound search. However, this enumeration scheme can only be used on small problem instances because the complexity of enumerating even a single ES policy is exponential in the number of activities in the input problem. A direct effect of this limitation is that there are very scarce computational results regarding ES policies in the literature. Our contribution in Chapter 4 is to describe some efficient less known enumeration schemes. We also extend the literature by proposing new enumeration schemes.

**On RQ3.2.** A solver procedure works by enumerating a large number of candidate policies. Measuring the robustness of enumerated policies along the way is typically done with simulation and constitutes a bottleneck in performance. As Leus observes in his recent work [44],

> " .. the only practical means to evaluate the objective function in stochastic scheduling is simulation, which is unfortunately quite time-consuming and makes the bulk of the running time of enumeration algorithms."

Simulating a policy can be computationaly expensive, especially when the problem includes multiple projects with many activities (e.g., NedTrain-specific instances). The number of candidate solutions considered within the availale time budget might be severely limited, which in turn limits effectiveness.

However, simulation is not the only option as far as ES policies are concerned because an ES policy can be treated as a stochastic PERT network. Sphisticated PERT network-specific methds can be applied instead. Estimating the statistical properties of activity start and completion times in a PERT network amounts to solving the stochastic PERT problem (described in detail in Chapter 3) and simulation is only one way to do this. This problem originates in Operations Research where it was proven to be intractable by Hagstrom [32]. Due to its applicability in various fields this problem has attracted the attention of diverese research communities. It was recently discovered solving this problem efficiently on networks of massive size (i.e., with up to millions of nodes) would have important implications in the industry of robust VLSI circuit design. This resulted in the research field of Statistical Timing Analysis (STA) of circuit designs [7, 27]. In STA a circuit design is represented as a *timing graph* which allows to capture and analyze variability of operation (e.g., due to manufacturing imperfections) in a statistical manner. Starting a few years back, the field of STA is continuously growing with well over 100 publications so far. The purpose of Chapter 5–*Statistical Timing Analysis*, is to indicate the mapping between ES policies and circuit timing graphs. Essentially, both ES policies and timing graphs constitute stochastic PERT networks. This enables the application of effient STA techniques (that range from analytical methods to sophisticated simulation methods) for measuring the robustness of an ES policy. In addition, Chapter 5 gives a detailed study of a simple and very efficient STA method, namely Linear-Gaussian STA (LG-SSTA). In large circuit designs this method has been reported to achieve a speed-up of up to 264x in comparison to simulation. This analytical method can be applied to Gaussian random activity durations with (optional) linear interdependencies. Effectively, Chapter 5 gives an answer to RQ3.2.

**On RQ3.3.** To make the search effective, policy enumeration should be guided by a heuristic that gives priority to those policies that will most likely have high robustness. Along with efficient policy enumeration and efficient robustness measurement, this is a third issue involved in the synthesis of a competitive search procedure for robust ES policies. Due to time limitations, RQ3.3 is not studied in this thesis and could constitute the subject of future work.

In Chapter 6 we gather some preliminary experimental results based on the methods presented in Chapters 4 and 5. Our experiments involve solving single and multi-project SRCPSP instances using the enumeration schema introduced in Chapter 4 and using both simulation and LG-SSTA for detecting good quality solutions. Our purpose is to investigate under which circumstances woult it be beneficial to use LG-SSTA instead of simulation. Preliminary evidence suggest that the use of LG-SSTA is beneficial for large instances with high durational variability. It should be noted that this is the case with NedTrain-specific problems. Finally, we conclude the thesis with Chapter 7 and propose research directions for future work.

# Chapter 2

# Deterministic Project Scheduling

## 2.1 Introduction

The context of this capter is as follows: given the data of a maintenance week we are asked to formulate a (deterministic) sceduling problem. The purpose is to solve it and obtain (if possible), a baseline schedule in which all projects finish on time. One objective of this Chapter is to explain how the data of a maintenance week can be encoded as a project scheduling problem. Another objective is to give an overview of previous work on solving such a problem. Yet another objective is to evaluate previous work against the problem description given in Section 1.2.

We start with a short high-level discussion on scheduling problems in genereal and justify our focus on *project* scheduling problems in particular. This section ends with an outline of this Chapter.

### 2.1.1 Project scheduling

Scheduling research consists of a variety of scheduling problem models and respective solution methods. An ontological overview of scheduling can be found in [63]. Different problem models are have been developed in order to better fulfil the requirements of different problem domains. In general, a scheduling problem consists of activities, resources, and constraints. For each model, a variety of both exact and heuristic solution procedures (solvers) have been developed. Given that most scheduling problems are intractable [8], heuristic procedures are mostly used for solving instances of practical size.

A distinction can be made between machine scheduling problems and project scheduling problems; the former being a special case of the latter. In machine scheduling we deal with jobs while in project scheduling we deal with activities. In project scheduling there are multiple types of resources and there are multiple units available of each resource type. Each activity requires the use of one or more units of one or more types of resources. However, in machine scheduling, resource constraints are simpler: there are multiple units available of only one type of resource: the machine. Each job requires the use of a single machine. Both types of problems include pair-wise precedence constraints that define a transitive, irreflexive relation on the set of activities (or jobs).

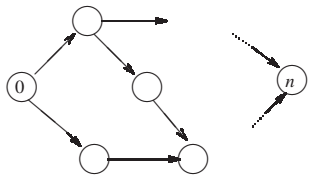


Figure 2.1: Example of dummy activities in a project plan.

Because of our need to model multiple resources with arbitrary capacities and arbitrary precedence constraints, this thesis focuses exclusively on project scheduling problems. The fundamental problem model in project scheduling is the Resource-Constrained Project Scheduling Problem (RCPSP). The decision version of the RCPSP asks whether there exists a schedule (i.e., an assignment of start-times to activities) that satisfies precedence and resource constraints and such that no activity completes later than a desired makespan. This problem is known to be NP-Hard (e.g., [22]).

### 2.1.2 Chapter outline

In Section 2.2 we give a formal treatment of the RCPSP and provide a set of extensions that enable the modeling of multiple projects with release/due-date constraints. In Section 2.3 we use these extensions to define the NedTrain Scheduling Problem (NSP): a problem model capable of representing an upcomming maintenance week. We also give an overview of prior work of our research group on fast heuristic solvers for the NSP.

## 2.2 The RCPSP

A RCPSP instance can be represented as a tuple $I = (P, Z, d)$. Element $P$ represents pairwise precedence constraints among activities the durations of which are described by vector $d$, while $Z$ represents resource constraints among activities and resources.

### 2.2.1 Precedence Constraints

Directed acyclic graph $P = (A, E)$, also known as a *plan*, encodes precedence constraints between a set of activities $A = \{0, ..., n\}$ with each edge $(a_1, a_2) \in E \subseteq A \times A$ signifying that $a_2$ cannot start unless $a_1$ has completed.

Figure 2.1 depicts an example plan. Dummy activities 0 and $n$ do not correspond to real workunits but have a special role: the start of the project corresponds to the completion of 0 while the completion of the project corresponds to the completion of $n$.[1]

Let us introduce the following relevant definitions with respect to plan $P = (A, E)$.

[1] Note that according to our notation, there are $n-1$ non-dummy activities in the project. In order to have $n$ non-dummy activities, some authors (e.g., [44]) define the set of activities as $A = \{0, \ldots, n+1\}$; i.e., such that $|A| = n+2$.

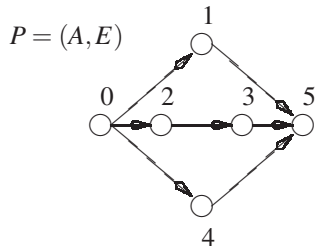


Figure 2.2: Example project plan with six activities.

.

**Definition 1.** *(Predecessors). The* predecessors *of activity $a_1 \in A$ is the set of activities from which there is a path to $a_2$, defined as*

$$pre_P(a_2) = \bigcup_{a_1 \in ipre_P(a_2)} (\{a_1\} \cup pre_P(a_1))$$

*where $ipre(a_2) = \{a_1 \in A \mid \exists (a_1, a_2) \in E\}$ is the set of* immediate predecessors *of $a_2$ and it consists of all activities that connect to $a_1$ with an edge.*

**Definition 2.** *(Successors). The* successors *of activity $a_2 \in A$ is the set of activities to which there is a path from $a_1$, defined as*

$$suc_P(a_1) = \bigcup_{a_2 \in isuc_P(a_1)} (\{a_2\} \cup suc_P(a_2))$$

*where $isuc(a_1) = \{a_2 \in A \mid \exists (a_1, a_2) \in E\}$ is the set of* immediate successors *of $a_1$.*

**Definition 3.** *(Anti-chains). The* anti-chains *of $P$ is the collection all mutually precedence-unrelated subsets of activities, defined as*

$$ant_P = \{X \in 2^A \mid \forall a_1, a_2 \in X : a_1 \notin (pre_P(a_2) \cup suc_P(a_2))\}$$

As an example, consider the plan $P = (A, E)$ of Figure 2.2, with $A = \{0,1,2,3,4,5\}$ and $E = \{(0,1),(0,2),(0,4),(1,5),(2,3),(3,5),(4,5)\}$. The set of anti-chains for $P$ is

$$\text{ant}_P = \{\{1,2\},\{1,3\},\{2,4\},\{3,4\},\{1,4\},\{1,2,4\},\{1,3,4\}\}$$

### 2.2.2 Resource Constraints

Tuple $Z = (R, \text{req}, \text{cap})$ encodes resource constraints between the set of activities $A$ and a set of resources $R = \{1, ..., k\}$. The number of units required by each activity from one or more resources is given by the function $\text{req} : A \to \mathbb{N}_+$. The maximum number of units available at any point in time by each resource (its *capacity*) is given by the function $\text{cap} : R \to \mathbb{N}_+$.

In addition to the plan of Figure 2.2, consider resource constraints $Z = (R, \text{cap}, \text{req})$ with resources $R = \{0,1\}$ with capacities $\text{cap}(0) = 3$, $\text{cap}(1) = 3$ and a specification for function $\text{req}(\cdot,\cdot)$ as given in Table 2.1.

| $a$ | req$(a,0)$ | req$(a,1)$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 2 | 1 |
| 2 | 0 | 1 |
| 3 | 0 | 2 |
| 4 | 2 | 2 |
| 5 | 0 | 0 |

Table 2.1: Resource constraints that associate activities in the plan of Figure 2.2 to resources $R = \{0,1\}$ with capacities cap$(0) = 3$ and cap$(1) = 3$.

When precedence constraints $P$ are combined with resource constraints $Z$ we may proceed with the following definitions.

**Definition 4.** *(Forbidden Sets). Consider some resource $r \in R$. The* forbidden-sets *of plan $P$ with respect to resource $r \in R$ can be defined as*

$$FS_P(r) = \{X \in ant_P \mid \sum_{a \in X} req(a,r) \geq cap(r)\}$$

*to contain those anti-chains of $P$ whose accumulated requirement of $r$ exceeds its capacity.*

To denote the collection of forbidden sets for all resources with respect to $P$ we write $\text{FS}_P = \cup_{r \in R} \text{FS}(r)$. Let us demonstrate this concept in our example $P$ and $Z$. The sets of forbidden sets with respect to $r_1 = 0$ and $r_2 = 1$ are

$$\text{FS}_P(r_1) = \{\{1,4\},\{1,2,4\},\{1,3,4\}\},$$
$$\text{FS}_P(r_2) = \{\{3,4\},\{1,2,4\},\{1,3,4\}\}$$

The set of all forbidden sets is then $\text{FS}_P = \text{FS}_P(r_1) \cup \text{FS}_P(r_2)$.

Stork and Uetz [66] have used the concept defined below to propose an alternative description of resource constraints in project scheduling.

**Definition 5.** *(Minimal Forbidden Sets). Consider plan $P$ and resource $r$. The set of* minimal forbidden sets *for $r$ is the set of all subset-minimal forbidden sets, defined as*

$$MFS_P(r) = \{X \in FS_P(r) \mid (X \setminus \{a\}) \notin FS_P(r), \forall a \in X\}$$

Stork and Uetz also propose a procedure for finding all minimal forbidden sets (with respect to constraints $P$ and $Z$). Note that the number of (minimal) forbidden sets might be exponential in the number of activities in $P$. Continuing our example, the set of MFSs for resources $r_1$ and $r_2$ are

$$\text{MFS}_P(r_1) = \{\{1,4\}\},$$
$$\text{MFS}_P(r_2) = \{\{1,2,4\},\{3,4\}\}$$

In Chapter 3 we shall refer back to forbidden sets, a concept which plays a central role in defining stochastic scheduling policies.

### 2.2.3 Schedules

In addition to precedence and resource constraints $P, Z$, let $d = (d_0, ..., d_n) \in \mathbb{N}_+^{n+1}$ associate each activity $a$ with duration $d_a$. The solution to RCPSP instance $I = (P, Z, d)$ is a schedule (an assignment of start-times to activities) that satisfies the given precedence and resource constraints. A schedule can be represented as a vector of start-times $s = (s_0, ..., s_n) \in \mathbb{N}_+^{n+1}$. The respective completion times are represented as a vector $c = s + d$ and the respective project completion time (the *makespan*) is then $c_n$.

**Definition 6.** *(Precedence feasible schedule). Consider precedence constraints $P = (A, E)$ and activity durations $d = (d_0, \ldots, d_n)$. A schedule $s \in \mathbb{N}_+^{n+1}$ is* precedence feasible *when $s_{a_2} \geq c_{a_1}$ for each $a_1, a_2 \in A$ such that $a_2 \in suc_P(a_1)$.*

Completing activities in the topological order defined by $P$ is known in scheduling literature as performing an earliest-start execution of the plan. This would result in a precedence-feasible schedule that can be defined as follows.

**Definition 7.** *(Earliest-start schedule). Consider plan $P$ and activity durations $d$. The earliest-start schedule $s$ can be computed by considering all $a_2 \in A$ in topological order and letting*

$$s_{a_2} = \max_{a_1 \in ipre(a_2)} c_{a_1} \qquad a_2 \neq 0, \tag{2.1}$$

$$c_{a_2} = s_{a_2} + d_{a_2} \tag{2.2}$$

*with $s_0 = 0$.*

The earliest-start execution simply involves considering activities in topological order and starting an activity as soon as it becomes eligible for execution. An activity becomes eligible for execution when all its immediate predecessors have completed, or at $t = 0$ if it has no predecessors.

Let us return to our example plan of Figure 2.2, and assume activity durations $d = (0, 2, 2, 3, 3, 0)$. The earliest-start schedule would be $s = (0, 0, 0, 2, 0, 5)$ and $c = (0, 2, 2, 5, 3, 5)$, with a makespan of $c_5 = 5$.

Continuing, let us define the profile of a resource with respect to a given schedule, as follows.

**Definition 8.** *(Resource profile). With respect to activities $A$ with durations $d$, resource constraints $Z$, and schedule $s$, the* resource profile *for resource $r$ is defined as*

$$prf(r, s, t) = \sum_{a \in ovp(s,t)} req(a, r)$$

*where $ovp(s,t) = \{a \in A \mid s_a \leq t \leq c_a\}$ is the set of activities overlapping at time $t$ according to $s$.*

In the scheduling literature, the profile of a resource is usually depicted as an extended Gantt chart (e.g., [44]) such as the one in Figure 2.3 which displays the resource profile of

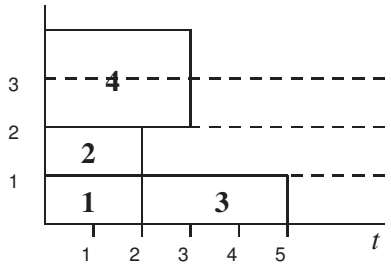


Figure 2.3: Profile of $r_2 = 1$ that corresponds to the earliest-start schedule ($s = (0,0,0,2,0,5)$ and $c = (0,2,2,5,3,5)$), for the plan in Figure 2.2.

$r_2 = 1$ that corresponds to the earliest-start schedule from our running example. Each box corresponds to an activity $a$; its length corresponds to $d_a$ and its height to $\text{req}(a,r)$. Note that dummy activities are not included in the profile depiction because they have a duratio of zero.

As can be seen in Figure 2.3, activities 1, 2, and 4 overlap in time. Note that these activities form a forbidden set. This violates the input resource constraints since the accumulated requirement of $r_2$ exceeds its capacity. Stated otherwise, the example earliest-start schedule is not resource-feasible according to the following definition.

**Definition 9.** *(Resource-feasible schedule). A schedule s is resource-feasible when* $prf(r,s,s_a) \leq cap(r)$ *for all* $a \in A$. *Alternatively, s is resource-feasible when* $ovp(s,t) \cap FS_P = \emptyset$ *for all* $t \in \mathbb{N}$.

Let $\lambda(P,d)$ and $\lambda(Z,d)$ each denote the (infinite) set of schedules that satisfy $P$, and $Z$, respectively. Now, let $\lambda(I) = \lambda(P,d) \cap \lambda(Z,d)$ denote the (infinite) set of feasible schedules for RCPSP instance $I$. We can now define the RCPSP as follows.

> **RCPSP.** For input $I = (P,Z,d)$, find a schedule $s \in \lambda(I)$ that minimizes project makespan $c_n$.

Let us close this Section with the following remark. With virtually unlimited resources (i.e., with $\text{FS}(r) = \emptyset$ for all $r$, or in the abscence of resource constraints), no real scheduling effort is required to find an optimal solution. The earliest-start schedule belongs to the set of optimal schedules.

### 2.2.4 Existing solution methods

As mentioned earlier, it has been proved (e.g., [8]) that the (decision version of) RCPSP is NP-hard. For a modern comprehensive review of variants and related solution methods, refer to [4]. Exact solution methods are mostly useful for small instances. These include branch-and-bound search strategies [53, 21, 10], mathematical programming [41], or even satisfiability problems [34, 2]. The largest part of RCPSP-related research focuses

on heuristic solution methods. Most popular heuristics construct a schedule by considering activities according to a given priority list [75, 40]. These methods are fast and are typically utilized by commercial products such as Microsoft Project. These approaches can be embedded in meta-heuristic procedures which according to the computational study of Hartmann and Kolisch [33] have been found to be the best performing.

For a review of RCPSP extensions such as with precedence constraints that enforce minimum/maximum time-lags between activities or non-renewable resources, refer to the work of Bruckner et al. [9] which also introduces a classification scheme, similar to the three-field scheme used in machine scheduling (introduced by Graham [31]). In what follows we shall propose extensions that enable the encoding of NedTrain-specific restrictions in the problem formulation.

### 2.2.5 Modeling release and due-date constraints

A train arrives at the depot at a specific release-date which introduces the constraint that its maintenance cannot start earlier than that. We can easily include this constraint to the RCPSP formulation, by letting the duration of the plan's source activity $d_0$ equal the release-date.

Furthermore, a train must be ready before its respective due-date. The due-date can be included as an extra parameter $q \in \mathbb{N}$ in the formulation; i.e., $I = (P, Z, d, q)$. But then the optimization objective must also be modified in order to account for this constraint.

A typical objective in the presence of a due-date is to minimize tardiness defined as $T = c_n - q$. Note, however, that this definition of tardiness rewards earliness; i.e., a schedule with negative tardiness will be prefered to one with zero tardiness. For various reasons earliness is often not desired in project management and an alternative objective for tardiness such as $T = \max(c_n - q, 0)$ is then used.

### 2.2.6 Modeling multiple projects

For simplicity, the plan $P = (A, E)$ has been so far assumed to represent a single project. NedTrain, however, is interested in the scheduling of maintenance on multiple trains at once. A multi-project plan can be used for this purpose, as shown in Figure 2.4.

The overall plan composes of $N$ sub-projects connected in parallel and they share a common source activity. The plan can be written as

$$P = \cup_{i=1}^{N} P_i \cup (\{0\}, E_0)$$

where $P_i = (A_i, E_i)$ is the $i$-th sub-plan and $E_0 = \{(0, a_i) \mid i = 1, ..., N\}$ where $a_i$ denotes the source activity of the $i$-th project.

Each sub-plan might be assigned its own release and due-date. In fact, this is the case with NedTrain maintenance projects. To model release-dates, each $d_{a_i}$ equals the release-date of the $i$-th project. To model due-dates, we include in the formulation a vector $q = (q_1, ..., q_n)$ where $q_a$ specifies a due-date for an activity $a \in A$ (not for a project). Now, due-dates of activities are set equal to the due-date of the project they belong to.

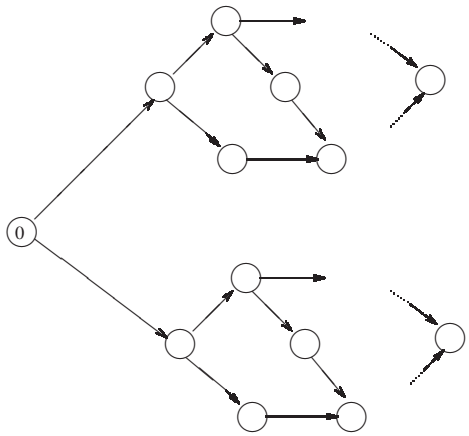


Figure 2.4: Example multi-project plan.

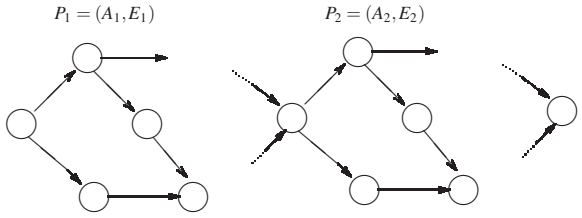


Figure 2.5: Project plan with two milestones.

The optimization objective will have to adjust to the presence of multiple projects. One common objective studied in the literature is minimization of the largest makespan, defined as $T = \max_{a \in M}\{c_a\}$ where $M$ contains the sink activities of the projects. Another common objective that is very relevant to the NedTrain case-study, is overall tardiness, defined as $T = \sum_{a \in M}\{T_a\}$ where $T_a = \max(c_a - q_a, 0)$. Yet another very useful objective is the overall number of tardy projects in which case $T_a$ equals 1 if $c_a - q_a \geq 0$ and 0 otherwise.

An RCPSP formulation with multiple projects is also known in the literature as the RCMPSP (e.g., [29, 76]) (for Multi-Project), which is at least as hard as the RCPSP.

### 2.2.7 Modeling multiple milestones

For management purposes, it is common practice to execute a project as a series of milestones. Milestones are easy to represent in an RCPSP formulation; they can be encoded as sub-projects connected in series. An example with two milestones is illustrated in Figure 2.5.

Every pair of consecutive milestones share a common activity; the sink of milestone $i$ is the source of milestone $i+1$.

As with multiple projects, each milestone may be assigned a respective due-date in a similar fashion. Multiple projects and multiple milestones can be combined in several interesting ways.

## 2.3 The NedTrain Scheduling Problem (NSP)

Based on the foregoing discussion we define here the NedTrain Maintenance Scheduling Problem (NSP): a problem model suitable for encoding the data of an upcomming NedTrain maintenance week.

An instance of the NSP is a tuple $I = (P, Z, d, q)$ with multiple projects and with release and due-date constraints modeled as we described in the previous sections. We state the problem as follows:

> **NSP.** For input $I = (P, Z, d, q)$, find and return $s \in \lambda(I)$ that minimizes tardiness $T = \sum_{a \in A} \max(q_a - c_a, 0)$.

Obviously, each project corresponds to a train. Due-dates are not specified per-project, but per-activity, with vector $q = (q_1, ..., q_n)$. The due-date of an activity is set equal to the due-date of the project it belong to. It is worth noting that removing release and due-date constraints from an NSP instance results in a RCMPSP instance.

Note that an alternative objective could be the minimization of the number of tardy projects. In fact, NedTrain receives an expensive penalty from NS proportional to the degree to which due-dates get violated. It would be perhaps more accurate to use as an objective the minimization of that penalty. However, the formula(e) for calculating the penalty as a function of due-date violations is not straightforward to use.

### 2.3.1 Existing work: Precedence Constraint Posting

Previous work for solving an NSP instance $I = (P, Z, d, q)$ was done by Evers [24] and by Wilson et al [73, 74]. Both used a problem formulation which is slightly different than the NSP. Precedence constraints and temporal constraints (i.e., release and due-dates) are encoded as a special type of Constraint Satisfaction Problem (CSP) [42], namely, a Simple Temporal Problem (STP) [20]. A STP includes a set of variables $\{t_1, ..., t_k\}$ (that correspond to timepoints) and a set of pair-wise constraints on those variables, each of which has the form $l_{ij} \leq t_j - t_i \leq h_{ij}$. The problem asks to find an assignment of values to timepoint variables such that all constraints are satisfied.

Figure 2.6a shows the plan of a project with four activities and with release date *rd* and due-date *dd*. Figure 2.6b shows an Simple Temporal Network (STN) encoding of the precedence constraints in the plan plus the release and due-date constraints. An STN is a convenient graph representation for a STP: nodes correspond to timepoint variable and edges to pair-wise constraints. A special timepoint variable $z$ that represents the start of time is introduced, along with constraint $z = 0$. For each maintenance activity $a$ two timepoint variables that correspond to its start and completion time are introduced, $s_a$ and $c_a$, respectively. Since it has duration $d_a$, a constraint $d_a \leq c_a - s_a \leq d_a$ is also introduced. If $a$ is the source activity of a project then a variable $x$ is introduced (instead of two variables as above). Assuming the project has release-date *rd* then also a constraint $rd \leq x - z \leq \infty$ is introduced. If $a$ is the sink activity of a project then a variable $y$ is introduced Assuming the project has due-date *dd* then also a constraint $-\infty \leq y - z \leq dd$ is introduced. Finally, for every

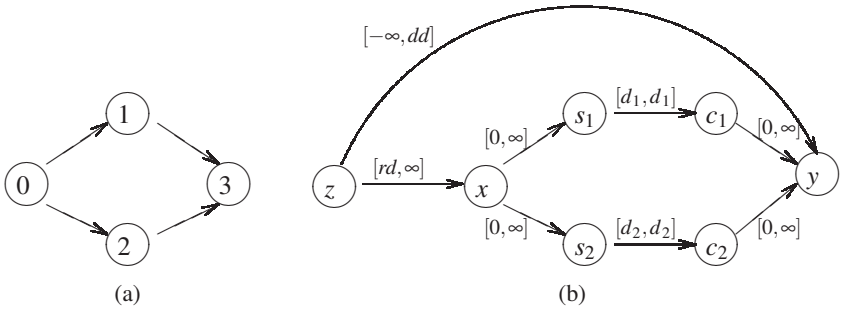


Figure 2.6: Example project (a) with a corresponding STN representation (b).

precedence constraint $(a_1, a_2)$ in the input project plan, an STP constraint $0 \leq s_{a_2} - c_{a_1} \leq \infty$ is introduced.

After the plan $P$ and release/due-dates have been STP-encoded, the purpose is to assign a value to timepoints without violating the constraints. From a time management point of view, we look for the smallest values possible that can be assigned and that satisfy all STP constraints. This is known as the *earliest-start* assignment and can be found in time polynomial to the number of STN nodes. An (earliest-start) assignment can be seen as (earliest-start) schedule $s \in \lambda(P,d)$ (i.e., that satisfies precedence constraints in the plan $P$). Note however that in the general case $s \notin \lambda(Z,d)$ (i.e., it does not satisfy resource constraints) since resource constraints were not taken into account for finding $s$. This means there must be at least one (resource,time value) pair $(r,t)$ such that $\text{prf}(r,s,t) > \text{cap}(r)$ (function prf was defined in Section 2.2). We say that pair $(r,t)$ identifies a *resource-conflict* between activities that use $r$ in parallel at timepoint $t$. To find a solution to $I$ (i.e., some $s \in \lambda(I)$) Evers enhance a Precedence Constraint Posting (PCP) algorithm named Earliest Start Time Assignment (ESTA) [13] that works as follows.

1. Let $P' \leftarrow P = (A,E)$.
2. Encode $P'$ as an STP and find earliest-start schedule $s$.
3. Check if there is a resource conflict in $s$.
4. If there are one or more resource conflicts, select one (let $(r,t)$ identify the selected conflict) and:
   - Select two activities $a_1, a_2 \in B$ where $B$ is the subset of activities that participate in the conflict.
   - Choose between precedence edges $(a_1, a_2)$ and $(a_2, a_1)$ and add it to $P'$.
   - Repeat from step 2.
5. If no resource conflict exists in $s$ then return $P'$ as a solution.

Thus, the ESTA algorithm repeatedly selects resource usage peaks that exceed resource availability and "irons them out" by "posting" extra precedence edges that limit allowable parallelism.

Note that the solution to $I$ is not schedule $s$ but $P'$; i.e., an extended version of input plan $P$ whose earliest-start schedule satisfies all constraints. This is actually a smart trick for handling uncertainty during project execution. Prior to the maintenance week, we formulate instance $I$ and find solution $P'$ with earliest-start schedule $s$ that we use in order to start activities throughout the week. During the maintenance week, suppose that the duration of activity $a$ turns out to be $d'_a > d_a$. We modify $I$ such that: (*i*) the duration of $a$ is now $d'_a$, and (*ii*) the plan $P$ is now $P'$. We solve $I$ to obtain a new $P'$ (which corresponds to a new, repaired schedule $s$).

The main point is that it will take much less time to solve the new version of $I$ because $P'$ already contains most necessary extra edges. This allows us to respond quickly to change. Another point is that if the repair takes place at time $t$, the repaired schedule up to time $t$ should be the same as the old schedule. This is guaranteed by the aforementioned process.[2] Furthermore, ESTA makes its selections (described earlier) according to heuristics for finding a solution $P'$ with optimal *flexibility*. The flexibility of some $P'$ is proportional to the number of possible solutions to its STP encoding. Intuitively, the more flexible $P'$ is, the easier it is to find a new solution after $d_a$ gets substituted with $d'_a > d_a$ (as described earlier). The run-time complexity of finding an earliest-start schedule for the STP encoding of $P'$ is $O(n^3)$ where $n$ is the number of timepoint variables. Wilson extends ESTA mostly by making it significantly faster. Instead of encoding $P$ as an STP he uses a much simpler representation which makes it possible to find the earliest-start schedule in $O(n)$. This cuts down significantly on the time required to react to change during project execution; i.e., to find a new flexible solution. This is particularly important for NedTrain-specific instances because they can be very large. Wilson also improves the flexibility heuristics.

### 2.3.2 Limitations of existing work

The rationale behind the NSP/PCP approach is that since most-likely durations are used to generate a flexible baseline schedule, a slightly deviant realized schedule will most-likely also satisfy the input due-dates. However it can be extremely unlikely that realized activity completion times will not exceed those in the baseline schedule. As a result, project management is not provided with reliable projections for activity start and completion times in order to assess and manage the risks associated with executing the project.

Furthermore, note that NedTrain deals with problem instances of industrial size. As a result, repairing the baseline schedule in an effective way during project execution might take a lot of time which pushes all maintenance projects closer to their due-date. Thus, NedTrain can only afford a few quick repairs of the baseline schedule during execution.

Another disadvantage stems from treating due-dates as hard temporal constraints. The baseline schedule might not be repaired to feasibility within the available amount of time (which can be very limited in comparison to the problem size). In fact, if one or more

[2] Note, that to avoid shop-floor nervousness, after time $t$, the repaired schedule should not deviate too much from the old schedule.

activities take much longer than expected, the baseline schedule might not be repairable at all regardless of the available amount of time, in which case project management will have to continue without the help of a scheduling system. Treating due-dates as hard constraints is not a realistic approach and in practice it should be expected that one or more projects will exceed their due-date.

## 2.4 Conclusion

In Chapter 3 we look at stochastic project scheduling as an alternative approach which enables us to incorporate the presence of uncertainty in the problem formulation. We extend the deterministic NSP model defined here to a stochastic variant in which uncertain durations are modeled as stochastic variables with known distributions and examine related solution methods.

# Chapter 3

# Stochastic Project Scheduling

## 3.1 Introduction

In Chapter 2 we introduced the NedTrain Scheduling Problem (NSP)–a multi-project extension of the RCPSP–an instance of which can represent the data of an upcomming maintenance week. The solution to such an instance is a baseline schedule that is feasible with respect to the input precedence and resource constraints, and in which all projects finish before their respective due-dates. In addition, we discussed prior work by Evers [24] and Wilson [73] for solving this problem with efficient Precedence-Constraint Posting heuristics. Finally, we explained how the unrealistic assumption that the exact duration of each maintenance activity is known in advance limits the effectiveness of this approach in the presence of uncertainty.

The purpose of this thesis is to develop a method that is more effective in ensuring the timely delivery of maintenance projects despite the effects of uncertainty. In this Chapter we switch to the research area of *stochastic scheduling*. More specifically, we introduce a stochastic extension of the NSP, namely the Stochastic NSP (SNSP). This problem model is identical to the (deterministic) NSP, with one exception: activity durations are no longer represented with a vector $d = (d_0, \ldots, d_n)$ (where of course, $n+1$ is the number of activities), but with a stochastic vector $D = (D_1, \ldots, D_n)$, the elements of which are stochastic variables with known distributions. Maintenance is a process that repeats on a weekly basis. Therefore, projects typically consist of standard and recurring types of maintenance activities. This enables the derivation of duration distributions by means of analyzing historical data, possibly in combination with expert judgement. Therefore, we assume that the duration of each type of activity is described with a probability distribution–a stochastic quantification of its variability. At this point, we make no assumption about the type of duration distributions.

Just as the NSP was an extension of the RCPSP, the SNSP is a NedTrain-specific extension of the Stochastic RCPSP (SRCPSP) [35, 49, 50, 25, 28, 65]. The solution to the SRCPSP (and to variants such as the SNSP) is no longer a baseline schedule, but a *scheduling policy*. We adopt the notion of scheduling policies as proposed by Möhring et al [49]. Roughly spoken, a scheduling policy $\Pi$ is a set of rules for taking scheduling decisions at

certain decision times $t$ and these decisions are based upon the observed past up to $t$, as well as the a-priori knowledge of the input data of the problem. Stated otherwise, a policy is a set of rules for determining when to start activities on-the-fly (while observing their outcome durations), making sure that the input precedence and resource constraints are not violated. Eventually this will lead to having executed the projet(s) according to a (feasible) schedule $s$ with completion times $c = s + d$. Of course, $d$ is the realization of stochastic vector $D$. But then, vectors $s$ and $c$ are realizations of stochastic vectors $S^{\Pi}$ and $C^{\Pi}$, respectively. We say that for a given policy $\Pi$, vectors $S^{\Pi}$ and $C^{\Pi}$ define a *stochastic schedule*.

The statistical characteristics of $D$ are given, which enables us to obtain $k$ samples of it. This, in turn, enables us to perform $k$ simulations of the process of carrying out the project(s) according to $\Pi$. We can therefore obtain $k$ samples for each element in $S^{\Pi}$ and $C^{\Pi}$. A sufficient number of simulations is the method typically used in stochastic scheduling for estimating the statistical properties (e.g., moments or the complete distribution) of the elements in $S^{\Pi}$ and/or in $C^{\Pi}$. The objective that is typically used in the SRCPSP is the minimization of $\mu(C_n^{\Pi})$ (where $\mu$ denotes the expectation operator). That is, the objective now becomes to find a policy that minimizes the project makespan by expectation.

### 3.1.1 Chapter outline

Section 3.2 gives a formal treatment of the SRCPSP, along with a quick-paced introduction to three essential classes of scheduling policies: Resource-Based (RB) policies, Activity-Based (AB) policies, and Earliest-Start (ES) policies. Furthermore, we discuss the issue of evaluating the quality of a scheduling policy by means of simulation.

Section 3.4 discusses stochastic PERT networks: directed acyclic graphs in which nodes correspond to activities with stochastic durations. In particular, we discuss a problem that was studied in the early years of stochastic scheduling research: given a PERT network the purpose is to compute the probability distribution of the completion time of a certain node. We explain how for the class of ES policies, evaluating the objective function amounts to solving this problem.

Section 3.5 introduces the Stochastic NedTrain Scheduling Problem (SNSP)–a multi-project extension of the SRCPSP. We also propose objective functions that aim at the minimization of risk.

## 3.2 The Stochastic RCPSP

A SRCPSP instance can be represented by a tuple $J = (P, Z, D)$. Parameters $P = (A, E)$ and $Z = (R, \text{req}, \text{cap})$ specify precedence and resource constraints as defined for the deterministic RCPSP in Section 2.2. Parameter $D = (D_0, \ldots, D_{|A|-1})$ associates each activity $a \in A$ with stochastic variable $D_a$ that describes its duration and the probability distribution of which is assumed to be known. It should be noted that independence between stochastic durations is not necessarily part of the model. The distribution of a duration can be estimated by analyzing historical data and by accounting for the most important sources of uncertainty that affect it; refer to [18] for a relevant methodology. The problem can be defined as follows:

**SRCPSP**. For input $J = (P, Z, D)$ find policy $\Pi$ that minimizes the expected value of the stochastic makespan, $\mu(C_n^{\Pi})$.

The solution to a SRCPSP instance is no longer a baseline schedule, but a *scheduling policy*. To better explain the concept of a policy, let us recall what happens during an earliest-start execution of the input plan $P$ (Section 2.2). The earliest-start execution simply involves considering activities in topological order and starting an activity as soon as it becomes eligible for execution. An (unstarted) activity becomes eligible at $t$ when all its immediate predecessors have completed, or at $t = 0$ if it has no predecessors.

Let $X(t) \subseteq A$ denote the set of eligible activities at $t$. We informally define a policy as an artifact $\Pi$ produced during the scheduling process for the following purpose: Upon $t = 0$ and at subsequent activity completion-times $t$, $\Pi$ used in combination with the observed past up to $t$ by some procedure in order to select a subset $Y(t) \subseteq X(t)$ of eligible activities to start at $t$, such that $Y(t) \notin \mathrm{FS}_P$ (i.e., it does not form a forbidden set). Executing the project activities according to $\Pi$ will lead to the realization of a feasible schedule defined by $s$ and $c = s + d$, where $d$ is the outcome activity durations (i.e., the realization of stochastic vector $D$).

Several classes of policies have been proposed in the literature. For an introduction, refer to Chapter 2 of [68] and to Chapter 9 of [22]. For a formal definition in the machine scheduling context and a characterization of various classes, refer to and to [49, 50].[1] Uetz [68] defines a policy as a dynamic process for deciding which activities to start at successive timepoints $t$, by observing the past up to $t$ and by accounting for the input problem parameters. According to the *non-anticipativity constraint* [25] the policy may not use information from the future, e.g., the actual realization of activity durations. A policy has been defined as a function

$$\Pi : \mathbb{N}_+^n \to \mathbb{N}_+^n, \qquad d \xrightarrow{\Pi} s$$

to denote that it maps a realization $d$ of $D$ to schedule $s = \Pi(d)$ and such that the stochastic schedule can be defined as $S^{\Pi} = \Pi(D)$ and $C^{\Pi} = S^{\Pi} + D$. However, there is another, combinatorial viewpoint on policies (e.g., [65, 68]) which is compatible with our own definition given earlier and is the one we shall use here.

### 3.2.1 The classes of RB, AB, and ES policies

A class of policies that is easy to comprehend is that of *Resource-Based (RB)* policies. A RB policy can be represented as a function $\Pi : A \to \mathbb{N}$ that assigns priorities to activities. Upon $t = 0$ and each subsequent activity completion time $t$, we consider all unstarted activities in the order of $\Pi$ and start them if this does not violate any precedence nor resource constraint. Stated otherwise, $Y(t)$ is composed by including as many activities as possible in the order defined by $\Pi$ (such that $Y(t) \notin \mathrm{FS}_P$). This class suffers from Graham anomalies [30], the most important being a possible increase in the project makespan due to decreasing activity

[1] To the best of the author's knowledge, no formal definition has been given in the context of project scheduling.

durations. Another class that does not suffer from the Graham anomalies is that of *Activity-Based (AB)* policies, introduced by Stork [65] as a member of the general class of *linear pre-selective* policies. An AB policy is also represented as an assignment of priorities to activities. However, in comparison to RB policies, there is an extra restriction in starting an eligible activity at $t$ (i.e., in composing $Y(t)$). An activity in $X(t)$ is only included in $Y(t)$ (i.e., started at $t$) if all activities with a higher priority have already completed. Due to this side-constraint AB policies do not suffer from Graham anomalies and appear to be the most appealing class of policies in the context of stochastic project scheduling. For recent competitive results refer to [6] and [5].

All remaining Chapters of this thesis study the class of *Early-Start (ES)* policies. This class was studied by Igelmund and Radermacher [35] and later by Stork [65] as a member of the general class of *pre-selective* policies. An ES policy can be represented as an extension of plan $P = (A, E)$ into $\Pi = (A, E \cup H)$ such that $\mathrm{FS}_\Pi = \emptyset$ and such that $\Pi$ remains acyclic. An ES policy can be constructed by adding to $P$ a set of extra precedence edges such that all forbidden sets are eliminated. The project is carried out simply by performing an earliest-start execution of $\Pi$. Upon $t = 0$ or subsequent activity completion times $t$ the set of eligible activities $X(t)$ will never form a forbidden set. Example solutions in form of ES policies are displayed in Appendix A.

Following Igelmund and Radermacher, Stork proposes a revised procedure for enumerating ES policies (in the solution-space of the input problem) by considering all sets of extra precedence edges that eliminate all minimal forbidden sets $\mathrm{MFS}_P$ (defined in Section 2.2). To eliminate a minimal forbidden set $F \in \mathrm{MFS}_P$ it suffices to add a precedence edge $(a_1, a_2) \in F^2$; i.e., between some pair of its members. Therefore, Stork's procedure involves considering each minimal forbidden set $F$ in some order and branching on all possible edges in $F^2$. Note that such a procedure requires a preliminary listing of all members in $\mathrm{MFS}_P$. Since the number of minimal forbidden sets might be exponential in the number of project activities, this procedure suffers from severe computational limitations. Stork concluded in his thesis that for instances of practical size one must resort to a computationally feasible class such as the class of AB policies. In Chapter 4 we give a detailed description of Stork's procedure. Furthermore, we extend the literature by proposing alternative, computationally feasible enumeration schemes that do not rely on a preliminary listing of all minimal forbidden sets.

### 3.2.2 Evaluating the objective function

The objective of SRCPSP is to find a policy $\Pi$ that minimizes $\mu(C_n^\Pi)$, the expected value of the stochastic project makespan. Simulation is commonly acknowledged as the only practical means to evaluate the objective function in stochastic scheduling. Evaluating the objective function by simulation is demonstrated in Table 3.1. The statistical characteristics of $D$ are given, which enables us to obtain $k$ samples of it. This, in turn, enables us to perform $k$ simulations of the process of carrying out the project(s) according to $\Pi$. We can therefore obtain $k$ samples of $C_n^\Pi$, from which we can estimate $\mu(C_n^\Pi)$.

The need for simulation in order to evaluate the objective function is generally considered a limitation of stochastic scheduling methods. As Leus points out in his recent work

| POLICYSIMULATION |
|---|
| • Input: Policy $\Pi$<br>• Input: Stochastic durations $D$<br>• Input: Number of simulations $k$<br>• Output: Expected value of stochastic makespan $\mu(C_n^\Pi)$ |
| 1. For $i = 1,2,...,k$<br>a) Obtain sample $d^{(i)}$ of $D$<br>b) Simulate the project execution according to $\Pi$ with durations $d^{(i)}$ and obtain samples $s^{(i)}$ and $c^{(i)} = s^{(i)} + d^{(i)}$<br>2. Estimate $\mu(C_n^\Pi)$ from samples $\{c_n^{(1)}, \ldots, c_n^{(k)}\}$. |

Table 3.1

[44]: ".. the only practical means to evaluate the objective function in stochastic scheduling is simulation, which is unfortunately quite time-consuming and makes up the bulk of the running time of enumeration algorithms". The number of necessary simulations for estimating the objective function with reliable accuracy is expected to increase with the number of project activities and also with the degree of durational variability.

For the particular class of ES policies, simulation is not the only means to evaluate the objective function. Project execution is carried out by an earliest-start execution of the ES policy. An ES policy can be treated as a so-called *stochastic PERT network* [47, 48]. Estimating some moment of $C_n^\Pi$ or even the complete distribution amounts to solving the well-known stochastic PERT problem, discussed in the following Section.

## 3.3 Existing solution methods

Most SRCPSP-related research amounts to a theoretical (rather than computational) study of classes. Igelmund and Radermacher study the family of preselective policy classes in [35], based on the concept of minimal forbidden sets (Section 2.2). A policy is preselective if it specifies a particular activity "that will have to wait", for each minimal forbidden set: regardless of what the realized activity durations turn out to be, this activity will not be started until the completion of at least one activity from the minimal forbidden set. We can demonstrate this concept further by looking at a member class of this family, namely the class of ES policies, introduced by Radermacher in [58]. As explained earlier, an ES policy can be represented by an extension $\Pi = (A, E \cup H)$ of the input plan $P = (A, E)$. For each minimal forbidden set $F \subseteq A$ in $P$, $H$ contains an edge $(a_1, a_2) \in F^2$. During the earliest start execution of $\Pi$, activity $a_2$ cannot start unless $a_1$ has completed. An exhaustive exploration

of all policies requires involves considering all minimal forbidden sets, the number of which can be exponential in the number of activities. To overcome this computational limitation, Möhring and Stork [51] introduced the family of linear preselective policies, a member of which is the class of AB policies explained earlier.

Stork [65] gathers computational experience for several classes of policies, by developing complete branch-and-bound search procedures. The class of Pre-Processing (PP) policies–a cross between ES and AB policies–was recently proposed by Ashtiani et al. in [5] along with a genetic algorithm. Ballestin et al. [6] propose a competitive genetic algorithm for the class of AB policies. In [4], Artigues et al. leverage the relation between so-called *resource flow networks* and ES policies. They describe a mathematical programming approach to find an ES policy that minimizes the maximum absolute regret in situations of complete uncertainty when activity durations cannot be associated with probability distributions. It is worth noting that their approach does not involve the consideration of all minimal forbidden sets and is therefore computationally feasible. This relation between resource flow networks and ES policies is analyzed in Chapter 4.

## 3.4 PERT Networks

A stochastic PERT network is a directed acyclic graph in which edges define pair-wise precedence constraints between activities with stochastic durations. A PERT network is sometimes refered to as a Stochastic Activity Networks (SAN). A PERT network (also known as a Stochastic Activity Network) is basically a mechanism for representing a project plan with uncertainty. The concept of PERT networks originates in the Program Evaluation and Review Technique (PERT) [47]: a statistical technique for measuring and forecasting progress in large research and development projects invented in 1958 on behalf of the Special Projects Office of the U.S. Navy for the POLARIS nuclear submarine missile programme.

Given a PERT network, the stochastic PERT problem asks to find the probability distribution of the project makespan. Related sub-problems that have been studied in Operations Research (OR) ask to find the mean and/or variance of the project makespan, the probability that a certain path becomes critical, or the probability that a certain activity will belong to a critical path. A large part of related reseach deals with activity durations which are discrete, independent random variables and no assumption is made about the type of their probability distribution. On the other hand, many results are based on particular distribution types such as exponential, normal, beta, or gamma distributions.

The literature on PERT-related problems is quite large and we shall only mention here a few select publications. See [1] as well as Chapter 9 of [22] for a systematic review. The complexity of the PERT problem was revealed by Hagstrom in [32]. She proved that (*a*) computing a value of the cumulative makespan distribution is #P-complete, (*b*) computing the mean is at least as hard, and that neither of (*a*) and (*b*) can be solved in time polynomial in the number of points in the range of the makespan unless P = NP. The difficulty of PERT-related problems is due to the exponential number of paths in the network but also due to reconverging paths that share one or more nodes which makes their respective lengths

dependent stochastic variables. Exact procedures for computing the makespan distribution have been proposed as early as 1965 by Martin [48], later by Fisher [26], also Hagstrom [32] and more recently by Schmidt et al. [60] and Connor [54]. Several methods for approximating the makespan distribution have been proposed, originally in the PERT specification by Malcolm [47], and then by e.g., Sculli [61], Dodin [23]. Also early work for bounding the makespan distribution includes that of Kleindorfer [38], or Robillard and Trahan [59] and a more recent computational study by Ludwig et al. [46]. Early work that concentrates on approximating the distribution by means of simulation includes that of Van Sylke [69], Burt and Garman [11], and Sigal et al. [62].

### 3.4.1 ES policies as PERT networks

The PERT technique was developed for managing uncertainty in large and complex military Resource and Development (R&D) projects. In those projects, resources are considered virtually unlimited. The primary concern is the timely project completion. Therefore, the application of the PERT technique in projects with scarce resources requires resolving conflicts with regard to resource demands between activities as they arise during project execution [72]. An ES policy can be treated as a so-called PERT network; in which the width of the partial order defined by the precedence edges is restricted to keep resource requirements within availability at all times during project execution. Stated otherwise, an ES policy is an extended image of the plan with extra precedence constraints that eliminate all forbidden sets. Therefore, no set of eligible activities during project execution will form a forbidden set. Having constructed an ES policy in the scheduling phase of project management enables the application of PERT-based management practices (e.g., [47, 12, 52]) to projects with limited resources (e.g., NedTrain projects).

Prior to project execution, the project management team may proceed with, e.g., an extensive simulation of the chosen policy.[2] The project management team may acquire information useful for risk management such as:

- The probability distribution of the completion time of one or more projects.
- The overal probability (i.e., the risk) of exceeding a certain due-date for one or more projects.

Furthermore, the project management team may acquire information useful for resource deployment such as:

- The probability distribution of the start and completion of one ore more project activities.
- The probability distribution of the earliest/latest time at which one or more resources should be available.

Furthermore, to evaluate the objective function it is necessary to estimate some of the moments of some of the elements in $C^{\Pi}$ (i.e., the stochastic completion times vector that

[2] Simulation is, of course, applicable to other classes of policies as well.

corresponds to a policy $\Pi$). Estimating the statistical properties of one or more completion times can also be used to design informed heuristics for guiding the construction of a policy, or for determining a dominance condition during, e.g., a branch-and-bound search. For instance, Stork [65] re-estimates the expected project makespan $\mu(C_n^{\Pi})$ when the current ES policy under construction is extended by yet another precedence edge. It should be noted that a few hundred simulations are used for this purpose. If this value is already higher than the expected makespan of the best policy constructed so far, the search backtracks since the expected makespan may only increase with the addition of more edges. Research on solution methods for the stochastic PERT problem presents one with a wide range of options beyond crude simulation. In fact, in Chapter 5 we leverage the PERT network nature of ES policies and propose efficient methods from the field of robust Very Large Scale Integration (VLSI) circuit design.

## 3.5 The Stochastic NSP

Based on our foregoing discussion we can now define the Stochastic NedTrain Scheduling Problem (SNSP): a stochastic variant of the NSP (defined in Chapter 2), suitable for encoding the data of an upcomming NedTrain maintenance week including an explicit representation of anticipated uncertainty by means of stochastic activity durations.

An instance of the (deterministic) NSP was defined in Chapter 2 as $I = (P, Z, d, q)$. An instance of the SNSP is defined as $J = (P, Z, D, q)$. We can map $I$ to a stochastic equivalent $J$ by replacing most-likely durations $d$ with stochastic durations $D$. The problem can be defined as follows:

> **Stochastic NSP**. For input $J = (P, Z, D, q)$ find policy $\Pi$ that minimizes $\mu(T) + \sqrt{\sigma^2(T)}$ where $T = \sum_{a \in A} \max(0, C_a^{\Pi} - q_a)$.

Motivated by the concept of $\beta$-robustness [17], we propose to not only minimize the objective function by expectation but also by variance. Only accounting for the expected value is more suitable for minimizing the average tardiness over multiple repetitions of the process of carrying out the same projects with the given policy. Our purpose, however, is to minimize the likelihood of having high tardiness over the single upcomming maintenance week. For this reason, a policy with a slightly higher mean tardiness but with a lower tardiness variance is preferable. From a project management point of view, lower variance means more predictability in perforamnce and therefore lower risk.

Accounting for variance might be especially important because of the nature of maintenance projects. Up until inspection activities have completed, the necessary repairs that must be performed on a train are generally unknown. The duration of certain activities that involve repair operations might therefore exhibit severe variability. If the completion time of one or more projects is directly affected by those activities, then the tardiness variance is expected to be high. We are thus interested in finding policies that prevent those activities from participating in the "critical path" of one or more projects. Estimating the variance in the objective function with sufficient accuracy might be significantly more computationally

expective than only estimating the mean. For this reason, variance is usually not taken into account in stochastic scheduling. One of our major contributions in this thesis is to propose, in Chapter 5, efficient methods for measuring both mean and variance in the objective function for the particular class of ES policies.

Continuing with out problem specification, to map $I$ into a stochastic equivalent $J$ by replacing most-likely durations $d$ with stochastic durations $D$ one must decide:

- whether $D_a$ are discrete or continuous,
- upon the probability distribution type of $D_a$ (i.e., shape),
- upon the parameters for the distribution of each $D_a$.

A working assumption for the remainder of this thesis is that $D_a$ are Gaussian random variables. One possible way to map $d$ to Gaussian $D$ is to let $\mu(D_a) = d_a$; i.e., let most-likely estimates be the mean values of stochastic durations. For a Gaussian duration with mean $\mu(D_a)$ it would be reasonable to assume a variability of 20% (i.e., $\sigma(D_a) = 0.2\mu(D_a)$).

## 3.6 Conclusion

In this Chapter we introduced the SRCPSP and some basic classes of stochastic scheduling policies. We continued with defining the SNSP as a multi-project, NedTrain-specific extension of the SRCPSP, effectively answering **RQ1**. In addition, we identified the class of ES policies to be attractive for use in the scheduling and control of NedTrain projects, effectively answering **RQ2**. We remind that an ES policy is a directed acyclic graph in which nodes represent activities. Since activities have stochastic durations, an ES policy can be treated as a so-called PERT network; the schedule may therefore be realized by completing activities in topological order. However, no activities will have to compete for the use of a resource while doing so. In contrast to a simple PERT network, the width of the partial order defined by the precedence edges in an ES policy is restricted such that at no point in time will resource requirement accumulate beyond availability. The class of ES policies is attractive from a project management point-of-view because it enables the application of standard PERT-based management practices (developed for handling uncertainty in projects with virtually unlimited resources) to projects with limited resources (as is the case with NedTrain projects).

The existing enumeration scheme for ES policies suffers from computational limitations and this renders the solution of problems of practical size infeasible. In Chapter 4 we extend the literature by proposing alternative, efficient enumeration schemes, effectively answering **RQ3.1**. In Chapter 5 we deal with the performance bottleneck that arises from the use of crude Monte Carlo simulation for evaluating the objective function, effectively answering **RQ3.2**. In particular, we leveraging the PERT network-like nature of ES policies. As an alternative to crude simulation, we propose the use of Statistical Static Timing Analysis (SSTA) from the domain of robust VLSI circuit design.

# Chapter 4

# Enumerating ES Policies

## 4.1 Introduction

In the previous Chapter we introduced the Stochastic NedTrain Maintenance Problem (SNSP; Section 3.5) as a NedTrain-specific extension of the Stochastic Resource-Constrained Project Scheduling Problem (SRCPSP; Section 3.2). More specifically, a SRCPSP instance is a tuple $J = (P, Z, D)$. Directed acyclic graph $P = (A, E)$ is the project plan; i.e., a set of activities $A = \{0, \ldots, n\}$ and their pair-wise precedence constraints $E \subseteq A^2$. Parameter $Z = (R, \text{cap}, \text{req})$ describes resource constraints in terms of (re-newable) resources $R = \{0, \ldots, k\}$. For each $r \in R$ there are $\text{cap}(r) \in \mathbb{N}_+$ units available, and each $a \in A$ requires $\text{req}(a, r) \in \mathbb{N}_+$ units. Uncertainty is quantified in the problem formulation by representing the durations of project activities with a vector of stochastic variables $D = (D_0, \ldots, D_n)$ where the duration $D_a$ of each $a \in A$ has a known probability distribution. With stochastic durations, a solution is no longer a deterministic schedule, but a policy, which may belong to one of the several classes of policies proposed in the literature.

Our objective in the rest of this thesis is to discuss solution methods for the class of Early-Start (ES) policies (Section 3.2.1). Our motivation is that ES policies enable the application of standard, PERT-based management practices (developed for handling uncertainty in projects with virtually unlimited resources) to NedTrain maintenance projects (with scarce resources). Just like the project plan $P = (A, E)$, an ES policy is a PERT project network $\Pi = (A, E \cup H)$ in which the width of the partial order has been further restricted (with the addition of $H$) such that resource requirement will never accumulate beyond availability when completing the project activities in topological order. To extend $P$ to ES policy $\Pi$ one must come-up with an extension $H$ such that no anti-chain of $\Pi$ forms a forbidden set according to the resource constraints (Section 2.2.2) and such that $\Pi$ contains no cycles. We shall hereafter denote with $\lambda_{P,Z}$ the solution-space of the input problem; i.e., $\Pi \in \lambda_{P,Z}$ iff $\text{FS}_\Pi = \emptyset$ and $\Pi$ is acyclic. For a visual impression of ES policies refer to the example of Appendix A.

The main issue with the class of ES policies is the lack of efficient solution methods that can solve SRCPSP instances of practical size. Our purpose, starting from this Chapter, is to extend the literature towards that direction. More specifically, our objective is to propose

a framework for the design of competitive solution methods for the SNSP. We distinguish between three components in this framework: (*i*) ES policy enumeration schemes for exploring $\lambda_{P,Z}$, (*ii*) heuristics for guiding enumeration towards good quality policies, and (*iii*) methods for evaluating an objective function. The subject of this Chapter is the enumeration of ES policies in $\lambda_{P,Z}$. We treat the subject of evaluating an objective function in the next Chapter and leave the study of heuristics for future work. To the best of the author's knowledge, the literature contains three methods for constructing ES policies. Two of the methods involve a constructive procedure and the third method invovles the solution of a mathematical programming problem.

The first method proposed by Stork [65]. He did a computational study on several classes of policies and proposed complete branch-and-bound search procedures for solving the SRCPSP to optimality (i.e., for finding a policy that minimizes the project makespan by expectation). To construct an ES policy, Stork extends $P = (A, E)$ to $\Pi = (A, E \cup H)$ by including in $H$ an edge between an arbitrary pair of members for every minimal forbidden set $F \in \text{MFS}_P$ (Section 2.2.2). If $\Pi$ is acyclic, then $\Pi \in \lambda_{P,Z}$. The advantage of this approach (over the other two) is that it enables a complete search in $\lambda_{P,Z}$. The major disadvantage is that constructing a member of $\lambda_{P,Z}$ might be infeasible for problems of practical size, since it involves considering every member of $\text{MFS}_P$, the size of which might be exponential in the number of activities [65, 66].

Another method was proposed by Leus and Herroelen [45] They develop a complete branch-and-bound search for enumerating all ES policies which are *compatible* with a given schedule $s$ (the compatibility of a policy with a given schedule will be explained later). Leus and Herroelen construct an ES policy by first constructing a so-called *resource flow network*, also known as a *resource allocation*. This is a weighted directed acyclic graph in which nodes correspond to activities. A flow network can be extracted from a schedule that is given as a solution to a deterministic RCPSP instance. Edges in the flow network describe how the units of a specific resource are passed between activities in the given schedule. The specifics of flow networks will be examined later in detail. Given SRCPSP instance $J = (P, Z, D)$, one can construct a deterministic instance $I = (P, Z, d)$ by choosing $d$ arbitrarily. What is important is that $I$ and $J$ have common precedence and resource constraints. Solving $I$ one can now obtain schedule $s$. For each resource included in the problem, one may derive from $s$ a flow network[1] by means of a constructive procedure the run-time of which is polynomial in the number of activities and the resource capacity. Adding the edges from the flow network of each resource to $P = (A, E)$ (ignoring the weights), one will obtain an ES policy $\Pi = (A, E \cup H)$. It has been proved that $\Pi$ belongs to $\lambda_{P,Z}$ [45]. This relationship between flow networks and ES policies is further analyzed in detail by Leus in his recent work [44].

Yet another method was proposed by Artigues et al. [4]. Using ideas from robustness optimization they developed a method for finding an ES policy that minimizes a maximum absolute regret objective (i.e., they do not solve the original SRCPSP, but a variant). A flow network is actually a graph representation of a so-called *resource flow*. The concept of a resource flow was introduced by Artigues et al. [3] as a solution to a mathematical

[1] In fact, multiple alternative flow networks can be extracted by the given schedule for a specific resource.

programming problem (i.e., it is an assignment to a set of variables) which can be formulated in terms of the resource constraints $Z$ of a (deterministic or stochastic) RCPSP problem. The specifics of this are also explained later in the Chapter. In [4] Artigues et al. derive the flow network of each resource (and thus, an ES policy) by first formulating and solving the aforementioned mathematical program to obtain a resource flow.

We extend the literature by proposing a method for creating an ES policy which involves constructing a so-called *Partial-Order Schedule* (POS). The concept of a POS was introduced by Policella [56, 55]. In Section 2.3.1 we described a Precedence-Constraint-Posting (PCP) approach for solving the RCPSP variant with maximum and minimum time lags between the start and completion of project activities. Remember that a solution is a Simple Temporal Network (STN) the earliest-start assignment of which defines a feasible schedule. A POS is also a STN solution to that same problem (i.e., a STN). However, *every* feasible assignment to a POS defines a feasible schedule. Policella describes a procedure for constructing a POS, given a feasible schedule. This procedure is similar to the one proposed by Artigues for constructing a flow network, given a feasible schedule. Just as with a flow network, an ES policy can be extracted from a POS. Adding the zero-lag temporal constraints from a POS as edges to the input plan $P = (A, E)$ results in an ES policy $\Pi = (A, E \cup H)$.

Inspired by Policella's work on constructing a POS from a given schedule, we propose a set of properties for directed acyclic graphs $\Pi = (A, E \cup H)$ that constitute an extension of the input plan $P = (A, E)$. We prove that if $\Pi$ has these properties, then $\mathrm{FS}_\Pi = \emptyset$ and also $\Pi$ is acyclic, hence $\Pi \in \lambda_{P,Z}$. Stated otherwise, that an extension of the plan that has these properties constitutes a solution to the input stochastic scheduling problem. This allows us to generalize the method of constructing an ES policy via flow networks or POSs. In particular, we propose a method for enumerating ES policies without the need to operate on a schedule. Therefore, the preliminary step of solving a deterministic scheduling problem is no longer necessary.

### 4.1.1 Chapter outline

Section 4.2 explains how to construct ES policies with Stork's method. Section 4.3 describes how to construct ES policies by constructing resource flow networks. Section 4.4 describes how to enumerate ES policies by constructing Partial Order Schedules. Last, Section 4.5 introduces a method that does not require the construction of a schedule as a preliminary step. Since the specifics of flow networks and POSs are not directly relevant to stochastic scheduling, the reader might choose to skip directly to the proposed ES policy enumeration method which does not involve the processing of a schedule. In fact, this is the enumeration scheme that we implement for our experiments in Chapter 7.

## 4.2 Eliminating Minimal Forbidden Sets

This Section describes the enumeration scheme aspect of a branch-and-bound procedure proposed by Stork in [65] for solving the SRCPSP with ES policies to optimality. Intuitively, a minimal forbidden set $F \in \mathrm{MFS}_P$ contains "just enough" activities that make it a

MFSELIMINATION

- Input: Plan $P$ and resource constraints $Z$
- Input: Set of MFSs
- Output: ES policy $\Pi$

1. Initialize $\Pi \leftarrow P$.
2. For each MFS $X$ in a certain order:
   a) Select edge $(a_1, a_2)$ from $X^2$.
   b) Add $(a_1, a_2)$ to $\Pi$.
3. Return $\Pi$

Table 4.1

forbidden set (i.e., such that $F \in \mathrm{FS}_P$). The principle behind this procedure is that $F$ can be "eliminated" by introducing an edge $(a_1, a_2)$ between an arbitrary pair of its member activities. Extending $P = (A, E)$ into $\Pi = (A, E \cup H)$ by including in $H$ one elimination edge for each $F \in \mathrm{MFS}_P$ results in having $\mathrm{FS}_\Pi = \emptyset$; i.e., all non-minimal forbidden sets are eliminated as well. A procedure for doing so is given in Table 4.1.

The complete enumeration procedure proposed by Stork branches on all choices in step 2.a and as such all possible solutions are enumerated; i.e., all $H$ that eliminate all forbidden sets are constructed. As such, this procedure can be used to solve $J$ to optimality. The main limitation of this procedure is computational since constructing a solution involves considering all minimal forbidden sets, the number of which might be exponential in the number of activities. A procedure for finding all minimal forbidden sets is proposed by Stork and Uetz in [66].

### 4.2.1 A practical example

This example is identical to the one in Section 2.2, different values are used for activity durations. Consider the plan $P = (A, E)$ of Figure 4.1. The anti-chains (Sections 2.2.1 and 2.2.2) of $P$ are

$$\mathrm{ant}_P = \{\{1,2\},\{1,3\},\{2,4\},\{3,4\},\{1,4\},\{1,2,4\},\{1,3,4\}\}$$

In addition, consider resource constraints $Z = (R, \mathrm{cap}, \mathrm{req})$ where $R = \{0, 1\}$ with capacities $\mathrm{cap}(0) = 3$ and $\mathrm{cap}(1) = 3$ and requirements req as defined in Table 4.2. The sets

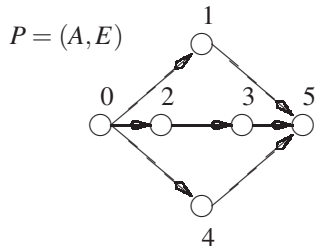


Figure 4.1: Example project with six activities.

.

| $a$ | req$(a,0)$ | req$(a,1)$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 2 | 1 |
| 2 | 0 | 1 |
| 3 | 0 | 2 |
| 4 | 2 | 2 |
| 5 | 0 | 0 |

Table 4.2: Resource constraints that associate activities in the plan of Figure 4.1 to resources $R = \{0,1\}$ with capacities cap$(0) = 3$ and cap$(1) = 3$.

of forbidden sets with respect to $r_1 = 0$ and $r_2 = 1$ are

$$\mathrm{FS}_P(r_1) = \{\{1,4\},\{1,2,4\},\{1,3,4\}\},$$
$$\mathrm{FS}_P(r_2) = \{\{3,4\},\{1,2,4\},\{1,3,4\}\}$$

The set of all forbidden sets for $P$ and $Z$ is $\mathrm{FS}_P = \mathrm{FS}_P(r_1) \cup \mathrm{FS}_P(r_2)$. The set of minimal forbidden sets for resource $r_1$ and $r_2$ are

$$\mathrm{MFS}_P(r_1) = \{\{1,4\}\},$$
$$\mathrm{MFS}_P(r_2) = \{\{1,2,4\},\{3,4\}\}$$

Extending $P$ with $H = \{(1,4),(3,4)\}$ yields a solution $\Pi = (A, E \cup H)$. Alternative edge-set extensions could be: $H' = \{(1,4),(4,3)\}$, $H'' = \{(1,2),(2,4),(3,4)\}$, and so on.

Figure 4.2 depicts part of the search-tree that Stork's procedure would explore if minimal forbidden sets are considered in the order: $\{1,2,4\}$, $\{1,4\}$, and $\{3,4\}$, Each node corresponds to a (partially-)constructed solution with the root node corresponding to $P = (A,E)$. Leaf nodes correspond to completely constructed solutions and in the depicted tree we can see the construction of $H$ and $H'$ discussed earlier. Note that adding edges $(1,4)$ and $(4,1)$ results to the introduction of a cycle and the respective node gets prunned.

This procedure assumes knowledge of the set of minimal forbidden sets, which can be computed with the method given in [66]. Generating a member of the solution-space involves considering all minimal forbidden sets in order. However, the number of all minimal

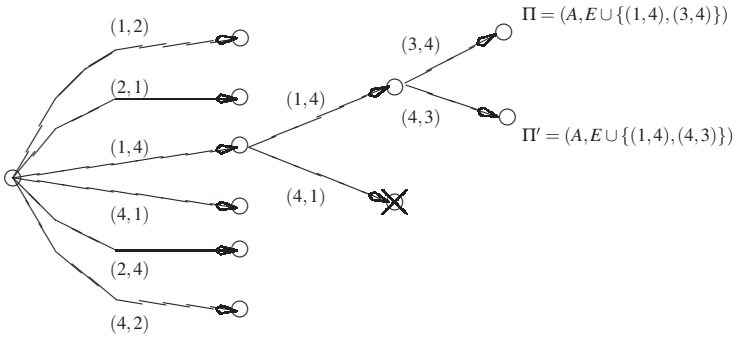


Figure 4.2: Example minimal forbidden set-based enumeration.

forbidden sets can be exponential in the number of activities. As a result, it is only possible to use this procedure for enumerating ES policies for small problem instances. Stork concludes in his thesis that for problems of practical size one should consider other classes of policies (e.g., AB policies).

## 4.3 Resource Flows

A resource flow [3] $f : A \times A \times R \to \mathbb{N}_+$ is a function that was proposed for modelling the flow of resource units between activities as it takes place in a project execution. In particular, $f(a_1, a_2, r)$ gives the amount of units from resource $r$ that are "passed" from $a_1$ to $a_2$ when the latter starts (at some point after $a_1$'s completion). As an example, a team of mechanical engineers that participate in carrying out activity $a_1$ start working on carrying out $a_2$ as soon as $a_1$ completes. Stated otherwise, a resource flow models a (partial) order in which activities access resources.

An adaptation of the definition of a feasible flow given in [45] to our notation follows.

**Definition 10.** *(Feasible resource flow). A resource flow $f$ is* feasible *with respect to resource constraints $Z$ when the following constraints are satisfied for every $r \in R$*

1. $\sum_{a_2 \in A \setminus \{0\}} f(0, a_2, r) \leq cap(r)$
2. $\sum_{a_1 \in A \setminus \{n\}} f(a_1, n, r) \leq \sum_{a_2 \in A \setminus \{0\}} f(0, a_2, r)$
3. $\sum_{a_1 \in A \setminus \{a_2, 0, n\}} f(a_1, a_2, r) = req(a_2, r)$
4. $\sum_{a_2 \in A \setminus \{a_1, 0, n\}} f(a_1, a_2, r) = req(a_1, r)$

Constraint 3 ensures that an activity is passed (by other activities) as many resource units as it actually requires according to $Z$. Constraint 4 ensures that, upon completion, an activity passes (to other activities) as many resource units as it uses. Constraint 1 ensures that the source of the project (i.e., activity 0) makes available for use (by non-dummy activities) as many resource units as there are available. Finally, constraint 2 ensures that no more than the available resource units are passed to the project sink.

In [45] Leus and Herroelen denote by $\phi(f)$ the precedence graph that corresponds to a flow $f$, defined as: $\phi(f) = (A,H)$ where $H \in A \times A$ is such that $(a_1,a_2) \in Y$ if and only if $f(a_1,a_2,r) > 0$ for at least one $r \in R$. For some $r \in R$, consider the weighted graph $(\phi(f),f)_r$ where $f(a_1,a_2,r)$ gives the weight of edge $(a_1,a_2)$. This is known as a resource flow network (e.g., [3]) for resource $r$. Note that the maximum cut of this network does not exceed $\text{cap}(r)$. Based on this, Leus and Herroelen [45] and Leus [44] argue that $H$ constitutes a sufficient selection with respect to resource constraints $Z$; i.e., $\text{FS}_{\phi(f)} = \cup_{r \in R}\text{FS}_{\phi(f)}(r) = \emptyset$.

Continuing, we adapt the definition given by Leus [45] for the compatibility of a flow with a plan $P$, as follows.

**Definition 11.** *(Compatible resource flow). Consider flow $f$ with $\phi(f) = (A,H)$. Flow $f$ is compatible with $P = (A,E)$ when $\Pi = P \cup \phi(f) = (A,E \cup H)$ is acyclic.*

Note that since $\text{FS}_{\phi(f)} = \emptyset$, also $\text{FS}_\Pi = \emptyset$ because $(E \cup H) \supseteq H$. Paraphrasing Theorem 1 of [44], if $f$ is both feasible with resource constraints $Z$ and compatible with plan $P$, then $\Pi = P \cup \phi(f) = (A,E \cup H)$ is an ES policy that constitutes a solution to all SRCPSP instances defined in terms of plan $P$ and resource constraints $Z$.

### 4.3.1 Efficient resource flow construction

For input RCPSP instance $I = (P,Z,d)$, in [3], Artiguess and Roubellat design a procedure that constructs flow $f$ that is feasible with $Z$ and compatible with $P$. The run-time complexity of this procedure is $O(|A|^2 \cdot |R| \cdot \max_{(a,r) \in A \times R}\text{req}(a,r))$. Artiguess and Roubellat embed the construction of a flow in the construction of a feasible schedule for $I$. However, as also noted by Deblaere [19], the flow construction can be decoupled from the schedule construction and can take place in a subsequent step, using as input the already constructed schedule.

The procedure for constructing resource flows is given in Table 4.3. By varying (e.g., randomizing) the choice made in step 3.a.i.A, multiple flows $f$ can be constructed; all feasible and compatible with respect to $P$ and $Z$.

### 4.3.2 Effient enumeration of ES policies

In light of the aforementioned equivalence between ES policies and resource flows (which is investigated in detail by Leus [44]), our observation is that this procedure can be used to enumerate a subset of the solution-space for an input SRCPSP instance $J = (P,Z,D)$ as follows:

- Choose arbitrary $d \in \mathbb{N}_+^n$ and construct a deterministic instance $I = (P,Z,d)$.
- Solve $I$ and obtain one or more schedules $s \in \lambda(I)$. For each schedule, apply the aforementioned procedure and construct one or more flows.
- For each flow $f$ obtain ES policy $\Pi = P \cup \phi(f)$.

Note that the choice of $d$ is irrelevant to the input $D$. This method allows the (incomplete) enumeration of ES policies in the solution-space of $J$ in polynomial time (per solution).

RESOURCEFLOW

- Input: Plan $P$ and resource constraints $Z$
- Input: Schedule $s \in \lambda(Z,d) \cap \lambda(P,d)$ for some arbitrary durations vector $d \in \mathbb{N}_+^{|A|}$
- Output: Resource flow $f$, feasible with respect to $Z$ and compatible with $P$

1. Initialize $f(a_1,a_2,r) \leftarrow 0$ for all $a_1,a_2 \in A, r \in R$
2. Initialize $f(0,n,r) \leftarrow \text{cap}(r)$ for all $r \in R$
3. For each activity $a_2 \in A$ in order of increasing $s_{a_2}$:
   a) For each resource $r \in R$:
      i. For $m \leftarrow 0, Q \leftarrow \emptyset$; until $m = \text{req}(a_2,r)$:
         A. Select $a_1 \in (A \setminus Q)$ : $c_{a_1} \leq s_{a_2}$
         B. $Q \leftarrow Q \cup \{a_1\}$
         C. $x \leftarrow \min(f(a_1,n,r), \text{req}(a_2,r))$
         D. $f(a_1,n,r) \leftarrow f(a_1,n,r) - x$
         E. $f(a_1,a_2,r) \leftarrow f(a_1,a_2,r) + x$
         F. $f(a_2,n,r) \leftarrow f(a_2,n,r) + x$
         G. $m \leftarrow m + x$
4. Return $f$

Table 4.3

### 4.3.3 A practical example

To better explain the operation of the flow construction procedure, we will first obtain a feasible schedule for the example RCPSP instance we saw earlier and then construct a resource flow. A resource flow is a modeling of the passing of resource units inbetween activities according to a schedule. As such, a convenient way to represent a flow is by means of a resource profile (e.g., [56, 55]), also known as an extended Gantt chart (Section 2.2). This chart is not only capable of depicting a schedule, but also a resource flow that could have been constructed by that schedule (recall that multiple flows can be constructed from a given schedule). In this chart, an activity is not only represented horizontally (i.e., its start and completion time on the time axis), but also vertically, to capture the amount of resource units it occuppies from a certain resource. As such, this type of chart cannot depict the whole flow $f(a_1,a_2,r)$ for all $r \in R$, but only for some specific $r \in R$.

Such a chart that depicts a feasible schedule for our example RCPSP instance and an (arbitrary) corresponding resource flow $f$ for resource $r_1 = 0$, is given in Figure 4.3. Dummy activities 0 and $n$ are not depicted since they do not use $r_1$ (or some other resource, in gen-

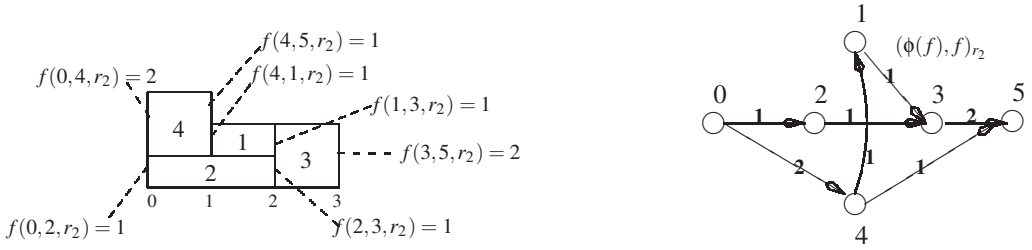


Figure 4.3: Extended Gantt chart representation of resource flow $f$ for $r_2 = 1$, and corresponding weighted flow network $(\phi(f), f)_{r_2}$.

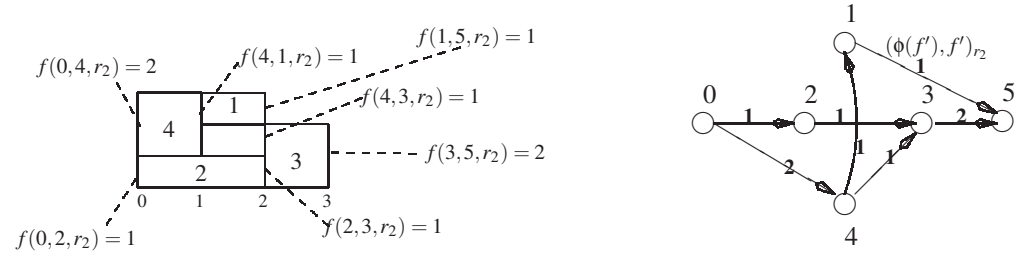


Figure 4.4: Extended Gantt chart representation of resource flow $f'$ for $r_2 = 1$, and corresponding weighted flow network $(\phi(f'), f')_{r_2}$.

eral). The operation of the flow construction procedure can be traced by visual inspection of the figure. Dashed lines point-out the passing of resource units between two activities. Next to the Gantt chart, there is a depiction of the weighted flow network representation of the resource flow, namely $(\phi(f), f)_{r_2}$. In Figure 4.4, an alternative flow $f'$ for the same schedule is depicted and its corresponding weighted flow network.

Flows $f$ and $f'$ can be mapped to ES policies $\Pi = P \cup \phi(f) = (A, E \cup \{(1,3),(4,1)\})$ and $\Pi' = P \cup \phi(f') = (A, E \cup \{(4,3),(4,1)\})$, respectively. Note that the maximum cut of neither of the flow networks exceeds $\text{cap}(r_2)$. As a result, policies $\Pi$ and $\Pi'$ eliminate all identified forbidden sets in $\text{FS}_P(r_2)$. Furthermore, according to the procedure, $(a_1, a_2) \in \phi(f)$ only if $s_{a_2} \geq c_{a_1}$ in the input schedule $s$. Since $s$ is precedence feasible, the constructed flow is guaranteed to be compatible with the plan $P$, which is reflected in our example since $\Pi$ and $\Pi'$ are both acyclic.

## 4.4 Chain-Form Partial Order Schedules

One of our initial observations in this thesis project was the relation between an ES policy and a so-called *Chain-form Partial Order Schedule* (POS), proposed by Policella in [56, 55]. Policella extended the PCP approach (a description of which was given in Section 2.3.1) into a two-step 'solve-and-robustify' process.

The first 'solve' step was described earlier and involves solving a variant of the RCPSP (the RCPCP/max). We remind the reader that the input plan is represented as a Simple

Temporal Network (STN) $S_0$ that encodes hard temporal constraints and the purpose is to find a resource feasible schedule that does not violate the temporal constraints. The input instance is solved by adding (zero-lag) precedence constraints to $S_0$, until its earliest-start solution corresponds to a resource and time-feasible schedule $s$. In the subsequent 'robustify' step $s$ is translated into a POS $S$. The POS is essentially a different STN, with the property that *every* feasible solution corresponds to a feasible schedule.[2] This way, a POS essentially encodes a set of solutions to the input instance, represented compactly as a STN. In this sense, a POS corresponds to a 'flexible schedule' that can absorb the effects of uncertainty. As activity start and completion times deviate from the ones in $s$, a new earliest-start $s'$ can be generated by constraint propagation through $S$.

### 4.4.1 The Chaining procedure

As Policella observe, a POS can be constructed by adding to the input plan extra (zero-lag) temporal constraints that eliminate all forbidden sets. Policella propose an efficient procedure called 'chaining' that seems to work by exactly the same principles as the aforementioned resource flow construction procedure. In fact, Deblaere [19] treat chaining as a type of procedure for constructing resource flow networks.

For input schedule $s$ (which, in Policella's work is an earliest-start solution of STN representation $S_0$ of input the plan), the chaining procedure is described in Table 4.4.

Each $r \in R$ is associated with $\text{cap}(r)$ chains of precedence-related activities. These chains are represented by variables $\text{head}(r,1),\ldots,\text{head}(r,\text{cap}(r))$ where $\text{head}(r,i)$ holds the head node of the $i$-th chain for $r$. These chains are initialized to empty. Each activity $a$ is visited in order of increasing start-time in $s$ and for every resource $r$ that it uses, $a$ will be appended at the end of exactly $\text{req}(a,r) \leq \text{cap}(r)$ of $r$'s chains. Activity $a$ can only be appended to those chains the head of which holds an activity with a completion time that is lower or equal to $a$'s start time (according to $s$). After all activities have been considered, the precedence edges of all constructed chains have been gathered in $H$. The precedence edges are then added as zero-lag temporal constraints into the original STN plan representation. When viewed as a simple precedence graph, the resulting POS is guaranteed to be void of forbidden sets with respect the input resource constraints. An illustrative example that better explains the idea behind chaining is given in Section 4.4.3.

### 4.4.2 Enumeration of ES policies

The similarities with the aforementioned resource flow costruction procedure are apparent. First, both procedures operate based on an input feasible schedule that constitutes a solution to a deterministic RCPSP problem. Second, both procedures use this schedule to find pair-wise precedence edges between activities that eliminate all forbidden sets. The flow construction procedure actually finds edge weights (i.e., $f(a_1,a_2,r)$) and non-zero weights map to precedence edges in $\phi(f) = (A,H)$. The chaining procedure, on the other hand, uses the input schedule to find an edge-set $H$ directly, without constructing a flow $f$.

[2] Furthermore, the earliest-start solution corresponds to the schedule that was used to construct the POS.

CHAINING

- Input: STN $S_0$ and resource constraints $Z$
- Input: Some feasible schedule $s$
- Output: Partial-Order Schedule $S$

1. Initialize $S \leftarrow S_0$
2. Initialize $\text{head}(r,i) \leftarrow \text{Nil}$ for all $r \in R, 1 \leq i \leq \text{cap}(r)$
3. For each activity $a_2 \in A$ in order of increasing $s_{a_2}$:
   a) For each resource $r \in R$:
      i. Let $\Gamma = \{1 \leq i \leq \text{cap}(r) \mid c_{\text{head}(r,i)} \leq s_{a_2}\}$
      ii. For $m \leftarrow 0, Q \leftarrow \emptyset$; until $m = \text{req}(a_2, r)$:
         A. $i \leftarrow$ SELECTCHAIN$(\Gamma \setminus Q)$; $Q \leftarrow Q \cup \{i\}$
         B. $a_1 \leftarrow \text{head}(r,i)$
         C. If $a_1 \neq \text{Nil}$ then add zero-lag edge $(a_1, a_2)$ to $S$
         D. $\text{head}(r,i) \leftarrow a_2$; $m \leftarrow m+1$
4. Return $S$

Table 4.4

For enumerating ES policies in the solution-space of SRCPSP instance $J = (P,Z,D)$, the chaining procedure can be used instead of the resource flow construction procedure, as follows:

- Choose arbitrary $d \in \mathbb{N}_+^n$ and construct a deterministic instance $I = (P,Z,d)$.
- Solve $I$ and obtain one or more schedules $s \in \lambda(I)$. For each schedule, use the chaining procedure to construct one or more edge-sets $H$ by varying the choice of step 3.ii.A.
- For each edge-set $H$ construct ES policy $\Pi = P \cup (A,H) = (A, E \cup H)$.

### 4.4.3 A practical example

To better explain the chaining procedure, we use the same schedule that we used in our previous example for explaining the construction of a resource flow. Figures 4.5 and 4.6. These figures illustrate two different resource profile representations based on the input schedule, specifically for resource $r_2 = 1$. Two different scenarios are examined with respect to the choice made in step 4.a.ii.A, upon visiting activity 1.

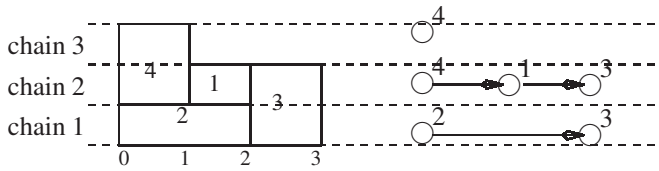


Figure 4.5: Example chaining of a schedule, resulting in edge-set $H = \{(4,1),(1,3),(2,3)\}$.

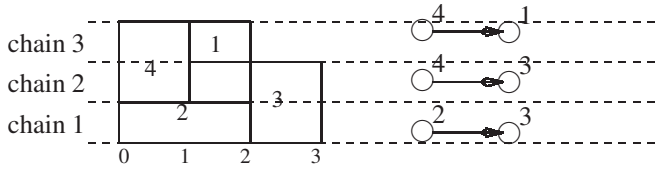


Figure 4.6: Example chaining of a schedule, resulting in edge-set $H = \{(4,1),(4,3),(2,3)\}$.

Choosing to put activity 1 in the second chain (Figure 4.5) results in ES policy $\Pi = (A, E \cup \{(4,1),(1,3),(2,3)\}) = (A, E \cup \{(4,1),(1,3)\})$. Choosing to put it in the third chain (Figure 4.6) results in $\Pi' = (A, E \cup \{(4,1),(4,3),(2,3)\} = (A, E \cup \{(4,1),(4,3)\})$. Note that all forbidden sets in $\mathrm{FS}_P(r_2)$ identified in Section 4.2.1 are eliminated for both these policies. In fact, this is the same pair of ES policies that we found using the resource flow construction procedure in Section 4.3.3.

## 4.5 Chain-Form Graph Construction

Motivated by the principles behind chaining we prove the sufficiency of a set of properties for precedene graph $\Pi$ such that it belongs in $\lambda_{P,Z}$. We propose a generalization of chaining, termed *chain-form graph construction* that does not rely on a feasible schedule in order to generate an ES policy solution. This procedure can also be used for generating POSs or resource flow networks.

We start with the following basic definition.

**Definition 12.** *(chain-form graph, chain). A chain-form graph is a precedence graph $C = \cup_{i=1}^{m} C_i$ where each $C_i = (U_i, H_i)$ is a chain. A chain $C_i = (U_i, H_i)$ on activities $U_i$ is a precedence graph with $H_i$ chosen such that $|H_i| = |U_i| - 1$ and $a_1 \in (suc_{C_i}(a_2) \cup pre_{C_i}(a_2))$ for all $a_1, a_2 \in U_i$.*

Note that by definition, all edges along a chain have the same orientation. Let us proceed with the following lemma. Consider chain-form graph $C = \cup_{i=1}^{m}(U_i, H_i) = (U, H)$ and subset of activities $X \subseteq U$. Furthermore, let $Y_a = \{1 \leq i \leq m \mid a \in U_i\}$.

**Lemma 1.** *$X \in ant_C$ iff $|\cup_{a \in X} Y_a| = \sum_{a \in X} |Y_a|$.*

*Proof.* If $|\cup_{a \in X} Y_a| = \sum_{a \in X} |Y_a|$ then every $a \in X$ is put in a unique set of chains. Otherwise, there is at least a pair $a_1, a_2, \in X$ for which $Y_{a_1} \cap Y_{a_2} \neq \emptyset$ and thus $X \notin \mathrm{ant}_C$. □

CFGCONSTRUCTION

- Input: Plan $P$ and resource constraints $Z$
- Input: Topological ordering $L$ of $P$
- Output: ES policy $\Pi \in \lambda_{P,Z}$

1. For each $r \in R$
   a) Construct feasible chain-form graph $C_r$ such that $L$ is compatible with $C_r$
2. Return $\Pi = P \cup (\cup_{r \in R} C_r)$

Table 4.5

Now consider resource constraints $Z = (R, \text{req}, \text{cap})$ that associate activities $A$ with resources $R$.

**Definition 13.** *(feasible chain-form graph). A chain-form graph $C_r = \cup_{i=1}^{m}(U_i, H_i) = (U, H)$ with $U \subseteq A$ is feasible with respect to $r \in R$ when:*

1. $|\{U_i \in \{U_1, \ldots, U_{cap}(r)\} \mid a \in U_i\}| \geq req(a,r)$ *for all $a \in U$; i.e., each $a \in U$ is put in at least $req(a,r)$ activity chains $(U_i, H_i)$*
2. $m \leq cap(r)$*; i.e., there are at most $cap(r)$ chains*

**Proposition 1.** *If $C_r$ is a feasible chain-form graph with respect to $r \in R$ then $FS_{C_r}(r) = \emptyset$.*

*Proof.* Consider subset of activities $X \subseteq U$. By Lemma 1 and Definition 13, $X \in \text{ant}_{C_r}$ iff

$$|\cup_{a \in X} Y_a| = \sum_{a \in X} |Y_a| \geq \sum_{a \in X} \text{req}(a,r)$$

By Definition 13, $|\cup_{a \in X} Y_a| \leq \text{cap}(r)$. But then $\sum_{a \in X} \text{req}(a,r) \leq \text{cap}(r)$ for all $X \in \text{ant}_{C_r}$ and thus, $\text{FS}_{C_r}(r) = \emptyset$. □

Consider the procedure given in Table 4.5.

**Proposition 2.** *Procedure* CFGCONSTRUCTION *returns a solution $\Pi \in \lambda_{P,Z}$.*

*Proof.* We must prove that $\Pi$ has the following properties: *i*) $\text{FS}_\Pi(r) = \emptyset$ for all $r \in R$, and *ii*) $\Pi$ is acyclic. For property *i* it suffices to observe that $\Pi$ can be seen as an extension of each individual $C_r$. Extending a partial order does not introduce new antichains. Thus, by Proposition 1, $\text{FS}_\Pi(r) \subseteq \text{FS}_{C_r}(r) = \emptyset$ for all $r \in R$. For property *ii* it suffices to observe that since $L$ is compatible with $P$ and with each $C_r$, it is also compatible with their union $\Pi$, which is therefore acyclic. □

CFGCONSTRUCTION-CHAINING

- Input: Plan $P$ and resource constraints $Z$
- Input: Some topological ordering $L$ of $P$
- Output: ES policy $\Pi \in \lambda_{P,Z}$

1. Initialize $\Pi \leftarrow P$
2. Initialize $\text{head}(r,i) \leftarrow \text{Nil}$ for all $r \in R, 1 \leq i \leq \text{cap}(r)$
3. For each activity $a_2 \in A$ in the order defined by $L$:
   a) For each resource $r \in R$:
      i. Let $\Gamma = \{1 \leq i \leq \text{cap}(r)\}$
      ii. For $m \leftarrow 0, Q \leftarrow \emptyset$; until $m = \text{req}(a_2, r)$:
         A. $i \leftarrow \text{SELECTCHAIN}(\Gamma \setminus Q)$; $Q \leftarrow Q \cup \{i\}$
         B. $a_1 \leftarrow \text{head}(r,i)$
         C. If $a_1 \neq \text{Nil}$, then add $(a_1, a_2)$ to $\Pi$
         D. $\text{head}(r,i) \leftarrow a_2$; $m \leftarrow m+1$
4. Return $\Pi$

Table 4.6

Note that for some $r \in R$ there are potentially multiple feasible chain-form graphs that can be constructed. By varying the construction of each $C_r$ in each of multiple calls to this procedure, one can enumerate at most as many different solutions in $\lambda_{P,Z}$. Furthermore, one might branch on the different choices for constructing each $C_r$ in order to explore (some part of) $\lambda_{P,Z}$ through a search-tree. Consideration of all minimal forbidden sets for $P$ and $Z$ is not necessary. All that is necessary is the construction of a feasible chain-form graph for every resource, which can be done efficiently.

**Chain-Form Graph construction with Chaining**

The behavior of CFGCONSTRUCTION can be implemented with a variation of the chaining procedure described earlier. It is of course not necessary to use an input baseline schedule. Instead, one can use a linear extension (i.e., a topological sorting) of $P$ for determining the order in which activities are visited. This procedure is given in Table 4.6.

# Chapter 5

# Evaluating the objective function

## 5.1 Introduction

In the previous chapters, we formulated the scheduling and control of NedTrain maintenance projects as a variant of the SRCPSP, namely the SNSP. We then took a first step towards the synthesis of solver procedures for solving instances of that problem for the class of ES policies. In the previous chapter, we discussed efficient procedures for constructing solutions. The subject of this chapter is the efficiency of evaluating an objective function on (partially) constructed solutions.

We remind the reader that evaluating an objective function requires mapping a policy $\Pi$ to a stochastic schedule; i.e., a pair of stochastic vectors $S^{\Pi}$ and $C^{\Pi}$ which correspond to stochastic activity start and completion times, respectively. Variable $S_a^{\Pi}$ ($C_a^{\Pi}$) is a stochastic modeling of what the start (completion) time of activity $a$ will be in the realized schedule after an (earliest-start) execution of the policy. For input $J = (P, Z, D)$, the SRCPSP asks to find a solution $\Pi \in \lambda_{P,Z}$ that minimizes $\mu(C_n^{\Pi})$–the expected project makespan. Since the SNSP is a multi-project extension of the SRCPSP, for simplicity we shall keep our focus on single-project SRCPSP instances.

The general consensus is that simulation is the only practical means to map a stochastic scheduling policy to a stochastic schedule. As Leus observes in his recent work on ES policies: "... the only practical means to evaluate the objective function in stochastic scheduling is simulation, which is unfortunately quite time-consuming and makes up the bulk of the running time of enumeration algorithms" [44]. Evaluating the objective function on every (partially) constructed solution can severely limit the number of enumerations, especially for real-world instances (such as instances of the SNSP) with many project activities.

It was explained in Chapter 3 that an ES policy defines a stochastic PERT network: a directed acyclic graph of precedence-related activities with random durations. As such, mapping a policy to a stochastic schedule amounts to solving the stochastic PERT problem, multiple variations of which have been studied in the literature. In the most basic version of this problem activity durations are discrete, independent random variables with given probability mass functions and the purpose is to find the cummulative distribution of the project makespan. Without an assumption about the type of duration distributions, the stochastic

PERT problem is rather generic. Even simplified versions of it, such as finding the expected project makespan (instead of the probability distribution) have been shown by Hagstrom to be intractable [32]. Exact procedures for computing the cummulative distribution of the project makespan have been proposed as early as 1965 by Martin [48], later by Hagstrom [32] and more recently by Connor [54].

The applicability of the stochastic PERT problem in multiple domains has attracted the attention of research communities beyond Operations Research. During the past decade, the industry of Very Large Scale Integration (VLSI) circuit design would see the rise of the discipline of Statistical Static Timing Analysis (SSTA). SSTA relies on the use of a *circuit timing graph* for modelling and analyzing the timing characteristics of a circuit design. In particular a timing graph is a stochastic model which captures the variability of timing characteristics caused by imperfections in the manufacturing process and the effects of the operating environment. Timing graphs are used for analyzing the robustness of a circuit design; i.e., the extent to which performance fluctuations can be absorbed such that operation remains within desired timing specifications.

The main concept behind the present chapter is that just as an ES policy, a timing graph essentially constitutes a stochastic PERT network. Throughout the last decade, the VLSI design industry has put a considerable amount of research effort (with well-over one hundred SSTA-related publications so far) on solving special cases of the stochastic PERT problem efficiently for networks of massive size (i.e., with up to millions of nodes). Starting in 2004, the simplest and most efficient SSTA technique, namely *Linear Gaussian SSTA*, was used for the development of *EinsStat*–the first SSTA-based VLSI design tool, which reportedly gave IBM a competitive advantage.[1] Since then multiple more accurate alternatives have been developed. Expecting the continuous advancement of SSTA techniques in order to follow the needs of the competitive VLSI industry, the present chapter can be summarized as an identification of the potential to use the momentum and results of SSTA research for the purposes of stochastic scheduling. Typically, stochastic PERT networks that arise in scheduling are many orders of magnitude smaller than those that represent VLSI circuits. However, the stochastic PERT problem must be solved at least once per enumerated ES policy. In fact, a well-informed heuristic search in the solution-space involves solving incremental revisions of a stochastic PERT problem for every (partially) constructed ES policy, making efficiency in doing so a primary concern.

### 5.1.1 Chapter outline

Section 5.2 explains the correspondence between In addition, this Section gives a high-level view of the SSTA research landscape. Taking a first step towards the proposed research direction, Section 5.3 explains the use of Linear Gaussian SSTA for the purposes of scheduling with ES policies.

[1] Article *Statistical timing can boost IC performance, panelists say* by Mike Santarini, Jun 2004, in `www.embedded.com`.

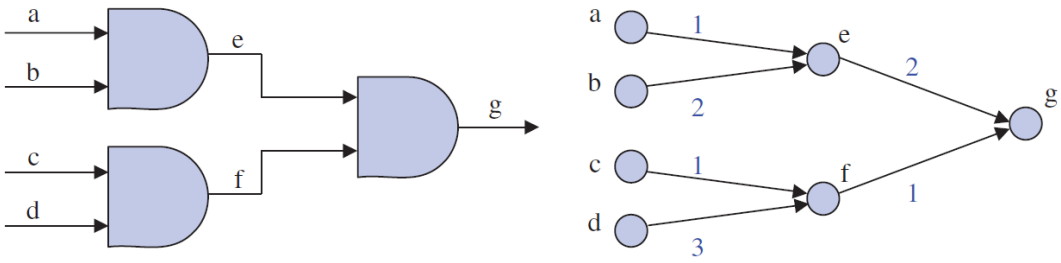


Figure 5.1: Sample circuit and corresponding timing graph (b).[27]

## 5.2 Statistical Static Timing Analysis

Variability in manufacturing leads to uncertainty in the behavior of circuit components. This makes analysis of timing characteristics a challenging task for nano-scale digital circuits. Traditionally, the timing characteristics of a circuit are modeled by means of a *timing graph* in which nodes correspond to components (e.g., logic gates, latches) and edges correspond to wiring. Every edge and node is associated with a delay; i.e., the time needed for a signal to propagate through.

Since the early 1990s, Static Timing Analysis (STA) has been the widely adopted method for analyzing the timing caracteristcs of a circuit design, by considering corner cases (e.g., best and worst case component delays). The basic STA analysis is conservative, in the sense that the delay of long paths in the circuit is overestimated, guaranteeing that the design will function at least as fast as predicted. As described in [71], with an increasing scale of integration STA becomes overly pessimistic and risky at the same time.

The limitations of STA and associated large costs of guard-banding VLSI circuits to ensure reliability has led to extensive research on Statistical STA (SSTA) (see [7, 27] for surveys). Delays are not treated as fixed numbers, but as random variables with known probability distributions. Our observation is that from an Operations Research (OR) point of view, an SSTA timing graph constitutes a stochastic PERT network. Signal arrival and exit times correspond to activity start and completion times and component delays correspond to activity durations. Figure 5.1, borrowed from [71], depicts an example circuit and its timing graph (where node *S* corresponds to the virtual sink component). To model within-die variations, the random delay of a component such as a logic gate must be represented as a function of individual random variables such as gate length, doping concentration, and metal thickness. Variability factors such as metal thickness might be shared among closely located components. As such, delays in the timing graph are correlated. Blaauuw [7] define the SSTA problem as that of computing the distribution of the overal delay between the arrival time of the virtual source component (typically set to zero) and the exit time at the virtual sink component. From an OR viewpoint, this is equivalent to computing the distribution of the project makespan (activity durations might be correlated).[2]

[2] The virtual source and sink components in a timing graph correspond to the dummy source and sink activity in a project plan.

Blaauw et al [7] and later Forzan et al [27] divide SSTA methods into those that rely on *i*) advanced Monte Carlo (MC) simulation, and *ii*) probabilistic analysis.

**Monte Carlo simulation.** MC-based SSTA involves performing thousands of full-scale circuit simulations by sampling the distributions of component delays. The distribution of an unknown timing quantity (i.e., a certain arrival or exit time) is then derived from the collected samples of that quantity. An advantage of MC-based methods is their generic nature which poses no restriction as to what type of probability distributions can be used to describe the timing quantities. The main disadvantage of MC-based SSTA is lack of efficiency, since a large number of simulations might be required to maintain the desired degree of accuracy. To achieve sufficient accuracy with as few simulations as possible, MC-based SSTA research focuses on *variance reduction* techniques such as *importance sampling* [67], *Critically Aware Latin Hybercube Sampling* (CALHS) [36], or *Quasi-Monte Carlo* (QMC) [70]. As pointed-out by Blaauw et al [7], another disadvantage of MC-based SSTA is that it might be difficult to perform incremental analysis after the designer has made a small change to the circuit.

**Probabilistic analysis.** Probabilistic analysis methods estimate timing quantity distributions by solving special cases of the stochastic PERT problem. We can distinguish between path-based and block-based methods. Path-based methods rely on performing a depth-first traveral of the timing graph. Path-based methods estimate the arrival time distribution of a certain node by *i*) analytical computation of the overal delay distribution along each path from the source to that node, and *ii*) analytical computation of the maximum over all paths. All path-based methods have a fundamental computational limitation because the number of paths is usually too large. To improve efficiency, some heuristic must be used to only consider a set of paths which are likely to become critical (e.g., [37]). Block-based methods rely on performing a breadth-first traveral of the timing graph. Upon visiting a node of the timing graph, the arrival time distribution is estimated by analytical computation of the maximum operator on the exit times of all immediate predecessors. The exit time distribution is then estimated by analytical computation of the sum of the arrival time plus the node's delay. According to Forzan et al [27], the most important block-based methods were proposed by Viswesvariah et al [71] and by Chang and Sapatnekar [15]. These two largely related methods are explained in detail in the following Section.

## 5.3 Linear Gaussian SSTA

Starting in 2003, one of the simplest and most efficient block-based SSTA methods–usually refered to as *Linear Gaussian SSTA* (LG-SSTA), resulted in the development of *EinsStat*, the first commercial SSTA-based VLSI design tool that reportedly gave IBM a competitive advantage.[3] Relevant research papers of key importance include Clark's results on the computation of the maximum over a set of Gaussian variables [16], the work of Chang

[3] Article *Statistical timing can boost IC performance, panelists say* by Mike Santarini, Jun 2004, in `www.embedded.com`.

and Sapatnekar on using these results to compose an efficient SSTA method [15], and the work of Viswesvariah et al on refining this SSTA method [71], which finally resulted in the development of EinsStat.

LG-SSTA relies on representing timing quantities (i.e., random node delays and random arrival and exit times), with a linear Gaussian model. We shall hereby describe the simple version of this model introduced by Chang and Sapatnekar instead of the slight variation introduced by Viswesvariah et al (known as the *canonical first-order model*). Our purpose is to examine this method from a stochastic scheduling point of view. Hence, we shall replace the terms random node delay, arrival time, and exit time, with the respective terms random activity duration, start time, and completion time.

### 5.3.1 The linear Gaussian model

The context of the following discussion is as follows. We are given (partially) constructed ES policy $\Pi$ along with random activity durations $D$ and our purpose is to map $\Pi$ to a stochastic schedule defined by random start times $S^{\Pi}$ and completion times $C^{\Pi}$.

According to the linear Gaussian model, the random duration $D_a$ of activity $a$ is expressed as:

$$D_a = \mu(D_a) + \sum_{i=1}^{m} \mathrm{cov}(D_a, i) X_i$$

(where $\mathrm{cov}(D_a, i)$ denotes the covariance between $D_a$ and $X_i$). That is, as a sum of so-called *Global Sources of Variation* (GSVs), which are assumed to be standard Gaussian; i.e., $X_i \sim N(0,1)$ for $i = 1, \ldots, m$. One implication is that as a sum of independent Gaussians, $D_a$ is also Gaussian. Another implication is the ability to capture (linear) dependencies inbetween durations. The number of GSVs and their covariances to durations can be chosen arbitrarily. Taken to one extreme, there is a single GSV correlated with every activity duration. Taken to another extreme, there can be as many GSVs as there are activities, and duration $D_a$ is only correlated to a respective GSV $X_a$; thus activity durations are independent. Note at this point, that the variance of $D_a$ is captured indirectly as

$$\sqrt{\sigma^2(D_a)} = \left( \sum_{i=1}^{m} \mathrm{cov}^2(D_a, i) \right)^{\frac{1}{2}}$$

A conclusion can now be driven about the distribution of the total duration $Y_p$ along a path $p$ of the network. As the sum of sums of independent Gaussians (i.e., individual durations), $Y_p$ is Gaussian. We remind the reader that in a PERT network, the start time of activity $a$ is defined as

$$S_a^{\Pi} = \max_{p \in \mathrm{path}(0,a)} Y_p$$

where $\mathrm{path}(0,a)$ contains all paths from source activity 0 to activity $a$. It should be noted at this point that random variables $Y_p$ are, in general, dependent because of the sharing of nodes between paths.

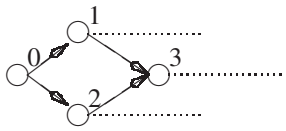


Figure 5.2: Example network fragment.

As Clark observes [16], the distribution of the maximum over a set of (dependent) Gaussians closely resembles a Gaussian bell but with a certain degree of skewness. This phenomenon is known in project management as the 'portfolio effect' (e.g., [43]) and we shall return to this later in the Chapter. The idea behind LG-SSTA is to compromise accuracy by ignoring the skewness and assuming that $S_a^\Pi$ (i.e., the result of the maximum operator) is Gaussian. Based on this, just as activity durations, (the unknown) start time can now be written in terms of sensitivies to GSVs according to the linear Gaussian model:

$$S_a^\Pi = \mu(S_a^\Pi) + \sum_{i=1}^{m} \mathrm{cov}(S_a^\Pi, i) X_i$$

Furthermore, we know that in a PERT network, $C_a^\Pi = S_a^\Pi + D_a$. According to the aforementioned simplifying assumption regarding the Gaussian nature of $S_a^\Pi$ we can write:

$$C_a^\Pi = \mu(C_a^\Pi) + \sum_{i=1}^{m} \mathrm{cov}(C_a^\Pi, i) X_i$$

What is now left is the computation of the expected value and the sensitivities of every $S_a^\Pi$ and $C_a^\Pi$ to the GSVs. This can be done by use of Clark's results [16], starting from the source activity and then propagating computations towards the sink node(s), in a topological order. This is explained with an example in the following Section.

### 5.3.2 Topological propagation of computations

Consider the example network fragment in Figure 5.2. Since activity 0 has no predecessors, $S_0^\Pi = D_0$. Thus, $\mu(S_0^\Pi) = \mu(D_0)$ and $\mathrm{cov}(S_0^\Pi, i) = \mathrm{cov}(D_0, i)$ for $i = 1, \ldots, m$. Furthermore, $C_0^\Pi = S_0^\Pi + D_0$. Thus, $\mu(C_0^\Pi) = \mu(S_0^\Pi) + \mu(S_0^\Pi)$ and $\mathrm{cov}(C_0^\Pi, i) = \mathrm{cov}(S_0^\Pi, i) + \mathrm{cov}(D_0, i)$ for $i = 1, \ldots, m$. In topological order, we then visit activities 1 and 2, for which $S_1^\Pi = C_0^\Pi$, $S_2^\Pi = C_0^\Pi$, and also $C_1^\Pi = S_1^\Pi + D_1$, $C_2^\Pi = S_2^\Pi + D_2$. The same computations can take place as for activity 0.

The interesting part of SSTA comes upon visting activity 3, for which $S_3^\Pi = \max(C_1^\Pi, C_2^\Pi)$. Computing $\mu(S_3^\Pi)$ amounts to solving the following problem:

> **GAUSS-MAX-MEAN**. For input Gaussians $A$ and $B$ with known $\mu(A), \mu(B), \sigma^2(A), \sigma^2(B)$, and correlation $\mathrm{cor}(A,B)$,[4] find $\mu(Z)$ where $Z = \max(A,B)$.

[4] $\mathrm{cor}(A,B) = \frac{1}{\sigma(A)\sigma(B)} \sum_{i=1}^{m} \mathrm{cov}(A,i)\mathrm{cov}(B,i)$

As early as in 1961 Clark [16] showed that:

$$\mu(Z) = \mu(A)\Phi(\zeta) + \mu(B)\Phi(-\zeta) + \chi\phi(\zeta)$$

where $\Phi, \phi$ are the Gaussian cdf and pdf, respectively, $\chi^2 = \sigma^2(A) + \sigma^2(B) - 2\sigma(A)\sigma(B)\text{cor}(A,B)$, and $\zeta = (1/\chi)(\mu(A) - \mu(B))$.

Computing $\text{cov}(S_3^\Pi, i)$ for $i = 1, \ldots, m$ involves solving the following problem:

> **GAUSS-MAX-COV**. For input Gaussians $A, B$, and $C^\Pi$ with known $\mu(A), \mu(B), \sigma(A), \sigma(B)$, correlation $\text{cor}(A,B)$, and $\text{cov}(A,X), \text{cov}(A,X)$, find $\text{cov}(Z,X)$ where $Z = \max(A,B)$.

Clark also showed that:

$$\text{cov}(Z,X) = \Phi(\zeta)\text{cov}(A,X) + \Phi(-\zeta)\text{cov}(B,X)$$

where $\zeta$ was defined previously. Visveswariah et al refer to the quantity $\Phi(\zeta)$ as the 'tightness probability'; intuitively interpreted as the probability that $A$ will be larger than $B$.[5]

Having computed $S_3^\Pi$, allows us to compute $C_3^\Pi = S_3^\Pi + D_3$ by simple addition. This concludes the computation of $S_3^\Pi$. If activity 3 had more than two immediate predecessors, then $S_3^\Pi$ would have to be computed as the maximum over more than two operands. As Clark demonstrates in his paper, this can be done with cascading computations, after observing that $\max(A,B,C) = \max(\max(A,B),C)$.

The computations demonstrated in the foregoing discussion can be propagated with a topological traversal throughout the rest of the network. The outline of such a procedure is given in Table 5.1.

Of course, step 2.a. may be skipped when $\text{ipre}_{a_2} = \emptyset$, or $S_{a_2}^\Pi$ can be set equal to $C_{a_1}^\Pi$ when $\text{ipre}_{a_2} = \{a_1\}$ (i.e., when $a_2$ has a single immediate predecessor). The run-time complexity of this procedure is clearly $O(n \cdot n \cdot m)$ where $n$ is the number of activities and $m$ is the number of GSVs.

### 5.3.3 LG-SSTA for solving the SRCPSP/ES

**Accuracy.** For Gaussian operands, the distribution shape of the maximum is actually skewed. Accuracy is mostly compromised in LG-SSTA due to ignoring this skewness, which allows the maximum to be treated as Gaussian. The error of this Gaussian approximation is larger when the operands have similar means but dissimilar variances [16].

Having found an ES policy solution to a sample SNSP instance, Figure 5.3 depicts the makespan distribution of a sample NedTrain maintenance project in the policy. Evaluating the robustness metric $\rho_2(C^\Pi) = -\mu(C_n^\Pi) - \sigma(C_n^\Pi)$ on the mean and variance obtained with Monte Carlo and with LG-SSTA we get a less than 8% difference. The makespan distribution estimated by simulation or by LG-SSTA during the search process does not need to be accurate. During the search process we are only interested in ranking alternative solutions according to their objective function score. That is, objective function evaluation must be accurate enough to preserve the "correct" ranking; i.e., if a large number of simulations

[5] Since $\Phi$ is the 'error function', note that $\Phi(-\zeta) = (1 - \Phi(\zeta))$.

LG-SSTA

- Input: ES policy $\Pi$
- Input: For each activity $a$, stochastic duration $D_a$ in form of:
  - Mean value $\mu(D_a)$
  - Covariance $\text{cov}(D_a, i)$ to GSV $X_i$ for $i = 1, \ldots, m$.
- Output: For each activity $a$, stochastic start time $S_a^\Pi$ and completion time $C_a^\Pi$ in form of:
  - Mean values $\mu(S_a^\Pi), \mu(C_a^\Pi)$
  - Covariances $\text{cov}(S_a^\Pi, i)$ and $\text{cov}(C_a^\Pi, i)$ to GSV $X_i$ for $i = 1, \ldots, m$.

1. Initialize the mean and sensitivies of $S_0^\Pi$ to zero.
2. For every activity $a_2$ in topological order:
   a) Compute $\mu(S_{a_2}^\Pi)$ and $\text{cov}(S_{a_2}^\Pi, i)$ for $i = 1, \ldots, m$ where $S_{a_2}^\Pi = \max_{a_1 \in \text{ipre}_\Pi(a_2)}\{C_{a_1}^\Pi\}$:
      i. If $\text{ipre}(a_2) = \emptyset$, then set mean and sensitivities of $S_{a_2}^\Pi$ to zero
      ii. If $\text{ipre}(a_2) = \{a_1\}$, then set $S_{a_2}^\Pi \leftarrow C_{a_1}^\Pi$
      iii. If $\text{ipre}(a_2) = \{a_1, a_2, \ldots\}$, then compute $\max_{a_1 \in \text{ipre}(a_2)}\{C_{a_1}\}$ in a cascading fashion, solving **GAUSS-MAX-MEAN** and **GAUSS-MAX-COV**.
   b) Compute $\mu(C_{a_2}^\Pi)$ and $\text{cov}(C_{a_2}^\Pi, i)$ for $i = 1, \ldots, m$ where $C_{a_2}^\Pi = S_{a_2}^\Pi + D_{a_2}$

Table 5.1

were used to evaluate the objective function. The degree of simulation accuracy can be chosen arbitrarily (by choosing the number of simulations). However, this is not possible with LG-SSTA.

**Efficiency.** The purpose of using LG-SSTA instead of crude simulation is to gain in efficiency. Simulation efficiency, however, is directly dependent on the number of simulation rounds. In the original SRCPSP formulation, the objective is to find a policy that minimizes the project makespan by expectation. Solely estimating the first makespan moment by simulation (on a (partially) constructed solution) might be significantly faster than also estimating the variance. With LG-SSTA it is not possible to avoid computing the variance. In fact, it is not possible to avoid computing the sensitivity of every activity to every GSV.

If the following conditions hold, a significant gain in efficiency by use of LG-SSTA should be noticed: *i*) a large number of simulation rounds is necessary to evaluate the ob-

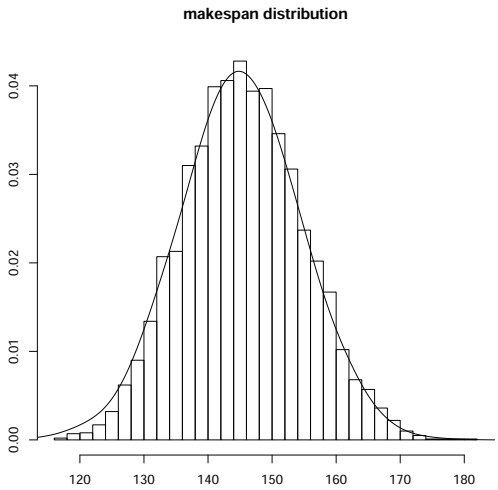


Figure 5.3: Sample makespan distribution: Monte Carlo (bars) vs. LG-SSTA (line).

jective function with sufficient accuracy, *ii*) variance must also be computed as part of the objective function.

**Correlated activity durations.** In the original SRCPSP formulation, an assumption is usually made that durations are independent random variables. However, as Leach argues in his study of the necessity for a stochastic modeling of activity duration uncertainty in real-life projects [43], such an assumption might be harmful. Recall that the project makespan can be expressed as the maximum over a set of paths from the source to the sink of the project. Leach observes that summing durations along a path without accounting for dependencies yields an overal path duration distribution that is far too steep with a far too narrow an uncertainty range. He concludes that for real-life projects, a correlation factor between activity durations must be applied. With LG-SSTA, duration correlations can be expressed in a fine-grained fashion, since every duration can be expressed as the linear combination of other durations.

# Chapter 6

# Experimental Results

## 6.1 Introduction

Throughout Chapters 3, 4, and 5 we discussed the SRCPSP and introduced the SNSP, a multi-project extension, for representing uncertainty in NedTrain maintenance projects. Further, we proposed efficient enumeration schemes for the class of ES policies, extending the literature. Simulation is acknowledged as the only practical means for evaluating common objective functions on enumerated policies. However, existing work suggests that simulation might constitute a severe bottleneck in the performance of enumerative solver procedures. Motivated by the PERT network nature of ES policies, we proposed the use of SSTA methods as an efficient yet accurate alternative to crude Monte Carlo simulation. Our contributions so far amount to a proposal for efficient enumeration methods and a proposal for the efficient yet accurate evaluation of the objective function. The synthesis of a competitive solver procedure requires yet another component: a heuristic for biasing the enumeration towards "high quality" solutions. However, we leave research on heuristics for future work.

This Chapter reports on experimental measurements that were obtained by implementing two "dummy" solver procedures. The procedure used in Experiment 1 (Section 6.2) generates solutions at random based on CFGCONSTRUCTION-CHAINING (Section 4.5). Stated otherwise, this procedure performs a random walk in the solution-space. This procedure is applied on a set of (single-project) SRCPSP instances and (multi-project) SNSP instances that were generated from the deterministic RCPSP test-set of Kolisch et al [39]. We report on the rate at which solutions can be enumerated by use of the aforementioned procedure. The focal point, however, is the use of LG-SSTA for evaluating the objective function, in comparison to the method of using different numbers of simulations. The objective that we study in this Chapter is the minimization of

$$\rho(C^{\Pi}) = \mu(T) + \sqrt{\sigma^2(T)}$$

where $T = \sum_{a \in A} \max(C_a^{\Pi} - q_a, 0)$ is the stochastic tardiness of an enumerated ES policy. Let us remind the reader that in a SRCPSP/SNSP instance, $A = \{0, 1, \ldots, n\}$ is the set of activities and that $C^{\Pi} = (C_0^{\Pi}, C_1^{\Pi}, \ldots, C_n^{\Pi})$ is their respective stochastic completion times.

In a SNSP instance, vector $q = (q_0, q_1, \ldots, q_n)$ represents their respective due-dates. In the original SRCPSP formulation there is only a single project and the objective is to minimize $\mu(C_n^{\Pi})$; i.e., the mean of the project completion time. In order to use the same objective function for both single- and multi-project instances, we simply assume a vector $q = (q_n)$ to be given for single-project instances. In our experiments we assume zero due-dates. Therefore, for the single-project instances the objective amounts to the mean-variance minimization of the project completion time. In multi-project instances, the objective amounts to minimizing the mean-variance of the sum of all project completion times.

To evaluate the objective function it is necessary to estimate the first two moments of the stochastic completion of each project. Making this estimation accurately with a large number of simulations might severly limit the number of enumerated solutions, in effect limiting overall performance. Since the time spent per simulation increases polynomially with the number of activities, this is especially true for large instances. NedTrain-specific instances might contain up to 30 maintenance projects each with anywhere between 5 and 30 activities. On the other hand, if too few simulations are used, good quality solutions might be overlooked due to inaccuracy. Therefore, the right balance must be striken between accuracy and efficiency when evaluating the objective function on candidate solutions. Instead of (crude) Monte Carlo simulation, Chapter 5 studied the potential to use an efficient yet accurate alternative, namely Linear Gaussian SSTA.[1] In Experiment 1 we investigate the question:

> Under which circumstances might LG-SSTA improve the effectiveness of an enumerative solver procedure?

The purpose of Experiment 2 (Section 6.3) investigates the same question for the particular case in which a heuristic is used to guide the enumeration of solutions. In particular, in case an estimation of the stochastic schedule takes place several times during the construction of a solution such that certain timing quantities in the stochastic schedule are used by the heuristic to make the remaining decisions. As mentioned earlier, our purpose is not to design a good heuristic for solving the SRCPSP and its extension for the class of ES policies. The heuristic that is used in Experiment 2 only serves the purpose of comparing the use of LG-SSTA and the use of simulation for this purpose.

## 6.2 Experiment 1

The purpose of this experiment is to examine if there is a gain in effectiveness when LG-SSTA is used instead of simulation in a solver procedure. In particular, we test the use of LG-SSTA against the use of simulations in order to determine whether an enumerated solution improves the best solution seen so far. Experiments 1.a and 1.b involve the same experimentation on problem instances with mild and medium durational variability, respectively.

[1] A limiting assumption for ensuring accuracy is that random durations are Gaussian (thus, continuous).

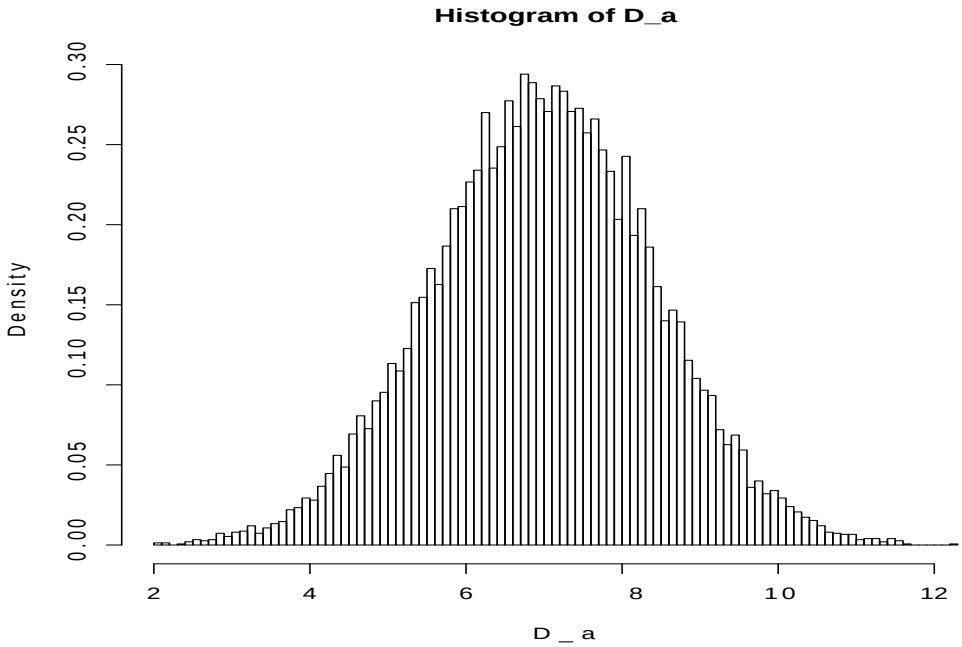


Figure 6.1: Histogram of an example Gaussian duration with a mean of 7 time units and a variability of 20%.

### 6.2.1 Experiment 1.a

A number of random solutions are enumerated within a specific time-budget by use of CFGCONSTRUCTION-CHAINING (Section 4.5; where SELECTCHAIN simply chooses at random). The examined objective is to minimize the tardiness mean and variance. We experiment with both single- and multi-project instances.

For single-project instances we use the well-known Kolisch [39] test-set. Since those are deterministic RCPSP instances, we map them to stochastic ones by converting each deterministic duration $d_a$ to a Gaussian duration $D_a$ with $\mu(D_a) = d_a$ and $\sigma(D_a) = 0.2\mu(D_a)$. A variability of 20% seems to be reasonable as can be seen in Figure 6.1 for a random duration with $\mu(D_a) = 7$. This test-set breaks down to groups `j30`, `j60`, `j90`, and `j120` where `j`$k$ contains 600 instances with $k$ activities. The instances in each group have different project plan structures and resource constraints and represent varying problem difficulty degrees (for the deterministic RCPSP case).

We only experiment with a select sample of about 40 instances from each group. Let us note that Balestin [6] also use the `j120` test-set to evaluate a proposed solution method for the class of AB policies. We shall not compare our results to theirs since our purpose is not to evaluate the effectiveness of a competitive solution method for the class of ES policies. For each of the `j30` and `j60` instances, a corresponding $N$-project instance is generated by including $N$ copies of the project plan and multiplying resource capacities by $N$. The source of each copy is connected to a global source node. Thus, the combined project plan contains a single source and $N$ sink nodes. Furthermore, each of the $N$ projects is assigned with a

random release-date.[2] We map j30 to groups 30x$N$ for $N = 5, 10, 25$ and j60 to group 60x$N$ for $N = 10$. The procedure for solving an instance is given in Table 6.1. For both single- and multi-project instances we set the due-dates of sink nodes to zero (and that of other nodes to infinity). Thus, for single-project instances the objective is to minimize the mean and variance of the project makespan while for multi-project instances it is to minimize the mean and variance of the sum of project makespans.

**Observations and conclusions**

Table 6.2 shows the rate of solution enumerations with random walk for each group of instances. Row STA corresponds to the use of LG-SSTA for evaluating the objective function (step 1.c of RANDOMWALK) and each row SIM-$i$ corresponds to using $i$ simulations. As a point of reference, row NONE shows the number of enumerated solutions per minute when not evaluating the objective function at all. One observation is that LG-SSTA is about as fast as 10 to 30 simulations and this analogy remains stable across all different groups of instances. We conclude that using more than, say, 20 simulations does not make sense since LG-SSTA is expected to be (much) more accurate and more efficient. It should be noted that for the chosen objective function, it is only necessary to estimate the mean and variance of the stochastic completion time of each project sink. Selective estimation of only specific timing quantities is possible when using simulation but not so when using LG-SSTA for which estimating the mean and variance of *all* start *and* completion times in the stochastic schedule is unavoidable.

Figure 6.2 shows the effectiveness of RANDOMWALK when using various simulation configurations for estimating the objective function, in comparison to using LG-SSTA. We use a time-budget of 10 minutes for solving each instance and each box-plot graph corresponds to a group of instances. Let $\rho^*_{\text{STA}}$ and $\rho^*_{SIM,i}$ denote the objective function score computed in step 2 of RANDOMWALK when using LG-SSTA and $i$ simulations, respectively, in step 1.c (i.e., the score of the best proposed solution). The vertical axis displays the deviation (in percentage) from $\rho^*_{\text{STA}}$ of $\rho^*_{\text{SIM,i}}$.[3] An interesting result is obtained: *using a single simulation consistently dominates every other method for proposing candidate solutions* (i.e., $\rho^*_{\text{SIM,1}}$ is the smallest most of the time).

Even though this is not visible in the provided graphs, it should be noted that the best solution proposed when using LG-SSTA is most of the time also proposed when using SIM-1. Furthermore, using a single simulation in step 1.c allows RANDOMWALK to enumerate a larger number of solutions.

Note that a single simulation results in variance being estimated as zero for all enumerated solutions. However, it is mostly the case that solutions with a lower tardiness mean value also have a lower tardiness variance *to an analogous degree*; i.e., that variance follows the mean value. Therefore, it appears to be affordable to ignore variance altogether;

[2] Thus, finding the optimal policy does not simply amount to finding an optimal policy for each project separately.

[3] In fact, this deviation is computed as the average over three individual repetitions. That is, each instance in a group is solved with RANDOMWALK using five different simulation configurations in step 1.c and using LG-SSTA (i.e., 6 times), and this repeats three times (i.e., 18 times in total). Thus, about 3 clock-hours have been spent per instance.

RANDOMWALK

- Input: SRCPSP/SNSP instance $J = (P, Z, D, q)$
- Input: Time-budget $\tau$
- Output: ES policy $\Pi \in \lambda_{P,Z}$ that minimizes $\rho(C^{\Pi})$ within time $\tau$

1. Until time-budget $\tau$ expires
   a) Generate random topological ordering $L$ of $P$
   b) Obtain $\Pi$ with CFGCONSTRUCTION-CHAINING-RANDOM
   c) If $\rho(C^{\Pi})$ decreases, propose $\Pi$ as a candidate solution
2. Return the proposed policy which minimizes $\rho(C^{\Pi})$, estimated accurately with 15000 simulations

CFGCONSTRUCTION-CHAINING-RANDOM

- Input: Plan $P$ and resource constraints $Z$
- Input: Some topological ordering $L$ of $P$
- Output: ES policy $\Pi \in \lambda_{P,Z}$

1. Initialize $\Pi \leftarrow P$
2. Initialize $\text{head}(r,i) \leftarrow \text{Nil}$ for all $r \in R, 1 \leq i \leq \text{cap}(r)$
3. For each activity $a_2 \in A$ in the order defined by $L$:
   a) For each resource $r \in R$:
      i. Let $\Gamma = \{1 \leq i \leq \text{cap}(r)\}$
      ii. For $m \leftarrow 0, Q \leftarrow \emptyset$; until $m = \text{req}(a_2, r)$:
         A. **Select** $i \in (\Gamma \setminus Q)$ **at random**; $Q \leftarrow Q \cup \{i\}$
         B. $a_1 \leftarrow \text{head}(r,i)$
         C. If $a_1 \neq \text{Nil}$, then add $(a_1, a_2)$ to $\Pi$
         D. $\text{head}(r,i) \leftarrow a_2$; $m \leftarrow m+1$
4. Return $\Pi$

Table 6.1

| Schedule estimation | j60 | j90 | j120 | 30x10 | 30x25 | 60x10 |
|---|---|---|---|---|---|---|
| NONE | $10^{6.1}$ | $10^{5.8}$ | $10^{5.7}$ | $10^{5.1}$ | $10^{4.3}$ | $10^{4.4}$ |
| STA | $10^{5.4}$ | $10^{5.1}$ | $10^{4.9}$ | $10^{4.0}$ | $10^{3.2}$ | $10^{3.4}$ |
| SIM-1 | $10^{5.7}$ | $10^{5.5}$ | $10^{5.3}$ | $10^{4.7}$ | $10^{3.8}$ | $10^{4.0}$ |
| SIM-5 | $10^{5.6}$ | $10^{5.4}$ | $10^{5.2}$ | $10^{4.5}$ | $10^{3.5}$ | $10^{3.8}$ |
| SIM-10 | $10^{5.5}$ | $10^{5.3}$ | $10^{5.1}$ | $10^{4.4}$ | $10^{3.3}$ | $10^{3.6}$ |
| SIM-30 | $10^{5.2}$ | $10^{5.0}$ | $10^{4.8}$ | $10^{4.0}$ | $10^{2.9}$ | $10^{3.2}$ |
| SIM-60 | $10^{5.0}$ | $10^{4.7}$ | $10^{4.6}$ | $10^{3.7}$ | $10^{2.6}$ | $10^{2.9}$ |
| SIM-100 | $10^{4.8}$ | $10^{4.5}$ | $10^{4.3}$ | $10^{3.5}$ | $10^{2.4}$ | $10^{2.7}$ |

Table 6.2: Policy enumerations per minute (RANDOMWALK).

i.e., minimizing the mean also minimizes variance most of the time. That is, assuming 20% Gaussian variability for all durations and especially for the single-project instances, it appears dominant to ignore the stochastic nature of the problem and treat it only in terms of an arbitrary realization of durations. Even though their study assumes uniform duration distributions, a similar observation is made for the j120 test-set by Balestin et al when duration variability is low.

Of course, a single simulation is not sufficient for accurately estimating the tardiness mean. In fact, we observe that the mean is usually underestimated. However, this underestimation seems to take place in an even fashion across candidate solutions. Thus, good solutions are still detectable. This phenomenon is easy to explain by looking at Figure 6.3 which displays the tardiness variability of proposed solutions for each group of instances (i.e., for each solution with tardiness $T$, we record the quantity $100(\sigma(T)/\mu(T))$). Even though durational variabilities in the problem instances are set to 20%, the tardiness variability of proposed solutions is much lower. It thus takes only a few simulations to evaluate the objective function (i.e., $\mu(T)+\sigma(T)$) with sufficient accuracy for a given solution.

### 6.2.2 Experiment 1.b

The previous results indicate that ignoring the stochastic nature of the problem at hand when ranking candidate solutions is a dominant choice. The use of a fast and accurate method such as LG-SSTA to substitute slow Monte Carlo simulation is therefore unnecessary. However, this result might be attributed to the low variability of the project durations. In fact, for the case of maintenance projects uncertainty is substantially greater. As explained in Chapter 1 about 50% of the time is usually spent on unplanned work revealed by inspections. An activity that involves the repair of one or more parts of the train is therefore expected to have a duration with high variability. The time spent per part can be characterized by a bimodal distribution with one peak around the origin and variance tending to zero and another peak around a non-zero mean with, say, 20% variability. The first and second peaks correspond to skipping the repair and not skipping the repair, respectively. Representing the sequence

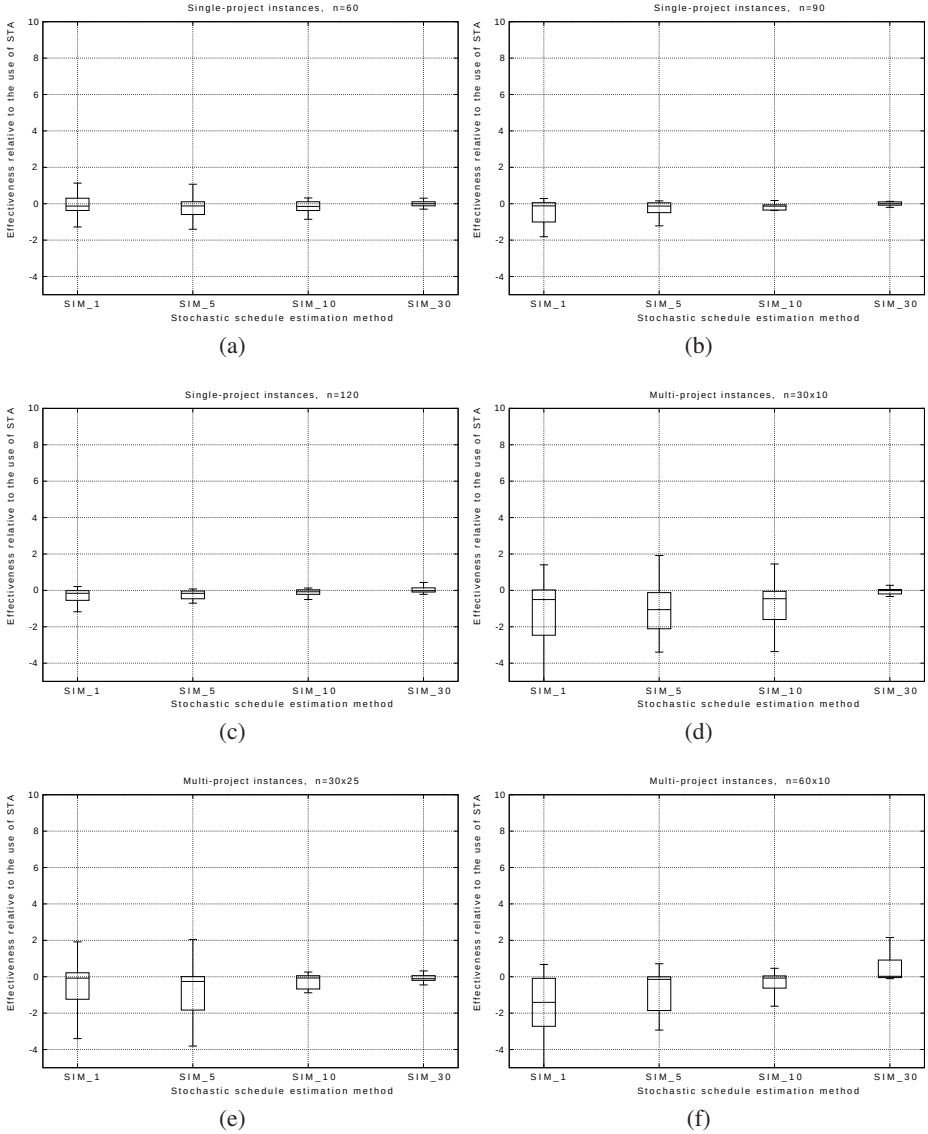


Figure 6.2: Effectiveness of RANDOMWALK across different methods for evaluating the objective function.

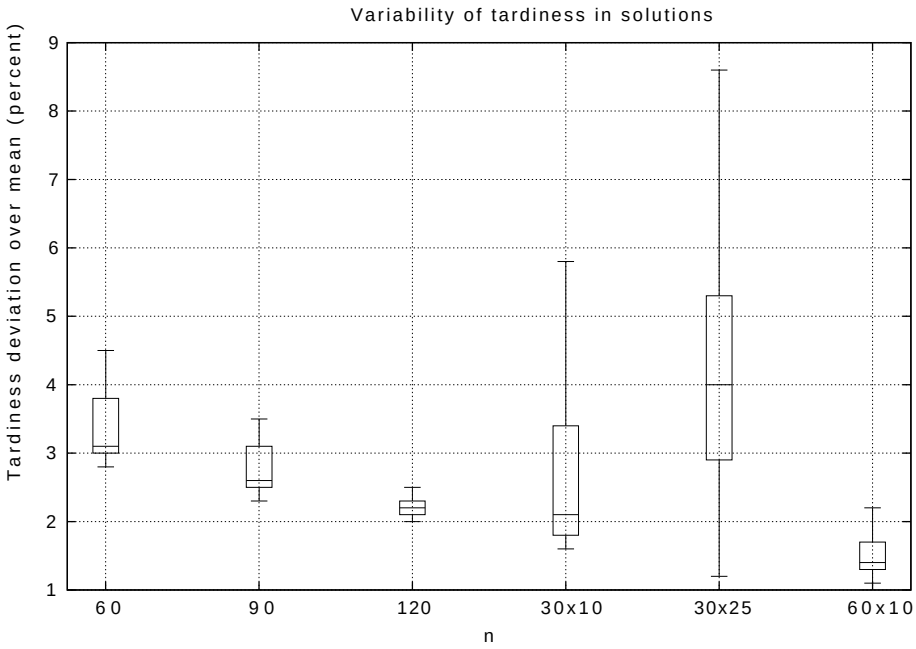


Figure 6.3: Tardiness variability.

of a few such part repairs as an activity results in a near-Gaussian duration distribution with a large mean and variability of about 30%.

Experiment 1.b is a repetition of Experiment 1.a but on instances that contain activities with substantial durational variability. In particular, we modified the previously described groups of instances. Up to 20% of the activities in each instance now have a duration mean of three times the original and a variability of 30%.

### Observations and conclusions

Figure 6.4 displays the new results. We observe that this time using 5 (instead of 1) simulations performs better than all other options. Furthermore, using only a single simulation performs at least as good as using LG-SSTA.

We can see that with larger variability, it is better to not ignore variance when computing the objective function in step 1.c. Otherwise, some of the better solutions are not proposed. However, the accuracy of LG-SSTA, even though only at the cost of about 20 simulations, is still an "overkill' in comparison to very few simulations. This phenomenon can be explained by looking at Figure 6.5. Again, even though certain activities are associated with a larger means and a larger variance, the variability of tardiness in proposed solutions remains very low. For instances with a much greater degree of durational variability, LG-SSTA might indeed be dominant over simulation.

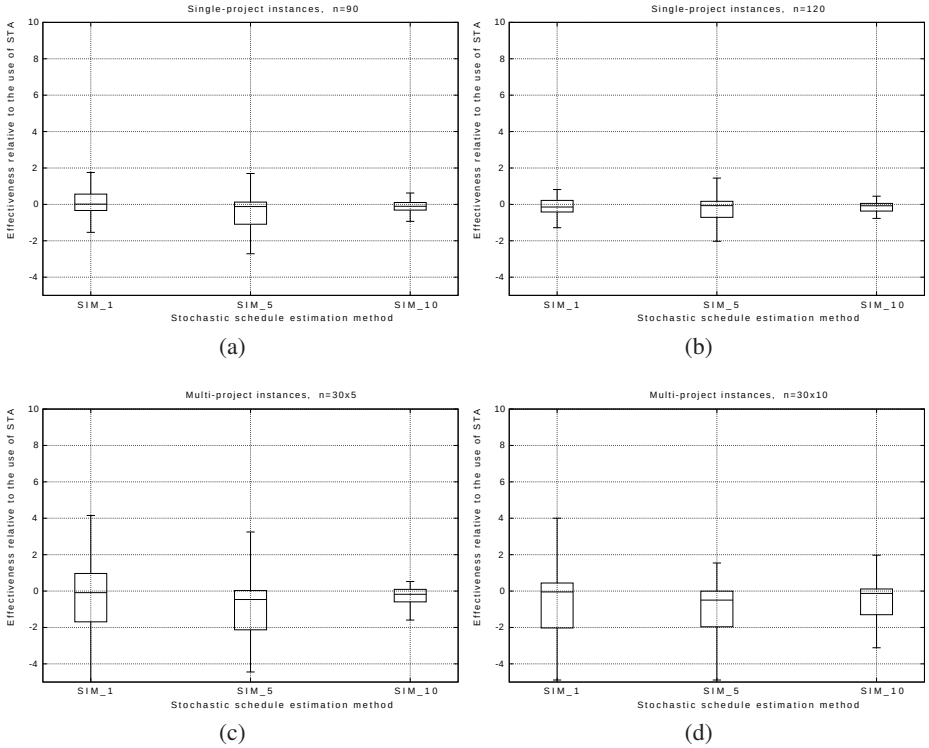


Figure 6.4: Effectiveness of RANDOMWALK across different methods for evaluating the objective function.

### 6.2.3 Concluding remarks

In this experiment, solutions were generated at random. The chains in which an activity is put (in the chain-form graph of each resource) are selected at random. For Experiment 1.a, deterministic single-problem instances were converted to multi and single-project stochastic instances by combination and by assuming a 20% durational variability. For Experiment 1.b, a new test-set was generated by associating at most 20% of the activities in each instance with a larger mean (three times the average) and a variability of 30%. This was done to emulate the high uncertainty with respect to the necessary repair tasks in the context of maintenance projects.

The use of LG-SSTA was tested against the use of a few simulations in order to determine whether an enumerated solution improves the best solution seen so far. For both instances with mild variability and those that emulate the higher uncertainty of repair tasks, the practice of using few simulations seems to be dominant since:

- the best solutions proposed when using LG-SSTA are not overlooked when using few simulations;

- the number of enumerated solutions is about an order of magnitude greater.

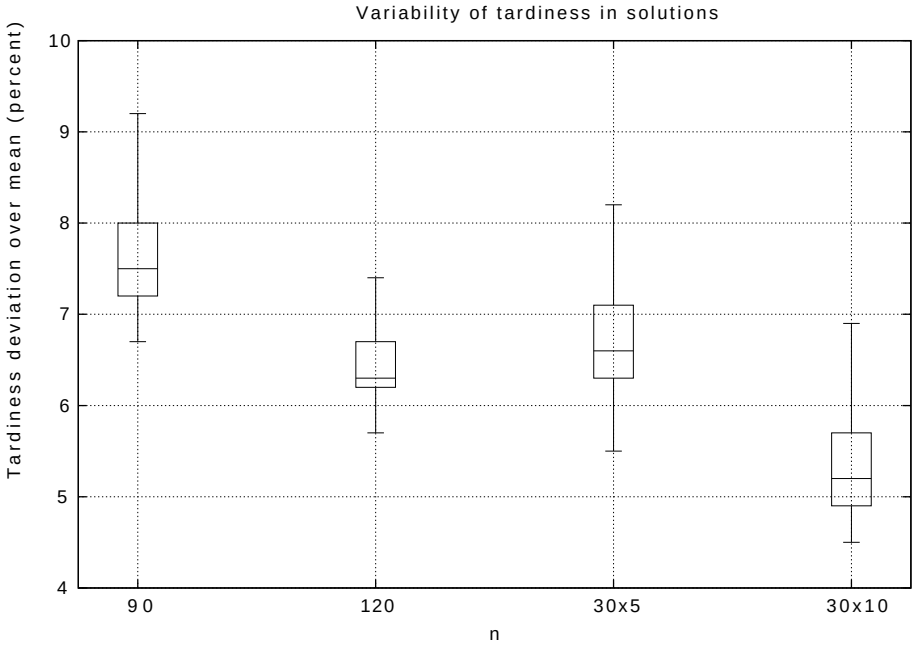


Figure 6.5: Tardiness variability in solutions to the new groups of instances, with activities that exhibit a higher degree of durational variability. In comparison to Figure 6.3, we observe a slight increase in the degree of variability.

A few simulations cannot estimate certain timing quantities of the stochastic schedule (that corresponds to an enumerated solution) with accuracy. However, compromising accuracy (for efficiency) seems to be sufficient for deciding whether a fully constructed solution "looks better" than another fully constructed solution. This phenomenon can be attributed to the low degree of tardiness variability in solutions (also for instances that emulate repair tasks uncertainty), as demonstrated in Figures 6.3,6.5.

In the following experiment we investigate whether using a few simulations is also accurate enough for making more subtle decisions, *during* the construction of a solution.

## 6.3 Experiment 2

We investigate the case of selecting the chains in which an activity is put with an informed heuristic. In particular when the heuristic relies on information provided by either LG-SSTA or simulation. Consider a procedure HEURISTICSEARCH whose operation is identical to that of RANDOMWALK except that now CFGCONSTRUCTION-CHAINING-HEURISTIC (Table 6.3) is used (instead of CFGCONSTRUCTION-CHAINING-RANDOM) to generate a solution.

This heuristic is based on a simple idea. An activity $a_2$ must be appended in $\text{req}(a_2, r)$ out of $\text{cap}(r)$ chains of the chain-form graph of resource $r$. The heuristic prefers those chains at the head of which is an activity whose *late completion* is the smallest. We define the late

CFGCONSTRUCTION-CHAINING-HEURISTIC

- Input: Plan $P$ and resource constraints $Z$
- Input: Some topological ordering $L$ of $P$
- Output: ES policy $\Pi \in \lambda_{P,Z}$

1. Initialize $\Pi \leftarrow P$
2. Initialize $\text{head}(r,i) \leftarrow \text{Nil}$ for all $r \in R, 1 \leq i \leq \text{cap}(r)$
3. For each activity $a_2 \in A$ in the order defined by $L$:
   a) Estimate the current stochastic completion times $C^{\Pi}$
   b) For each resource $r \in R$:
      i. Let $\Gamma = \{1 \leq i \leq \text{cap}(r)\}$
      ii. For $m \leftarrow 0, Q \leftarrow \emptyset$; until $m = \text{req}(a_2, r)$:
         A. **Select** $i \in (\Gamma \setminus Q)$ **that minimizes** $\mu(C^{\Pi}_{a_1}) + \omega\sigma(C^{\Pi}_{a_1})$**, where** $a_1 =$ **head**$(r,i)$
         B. $Q \leftarrow Q \cup \{i\}$
         C. $a_1 \leftarrow \text{head}(r,i)$
         D. If $a_1 \neq \text{Nil}$, then add $(a_1, a_2)$ to $\Pi$
         E. $\text{head}(r,i) \leftarrow a_2$; $m \leftarrow m+1$
4. Return $\Pi$

Table 6.3

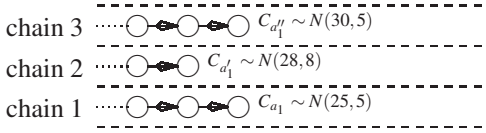


Figure 6.6: Example chain-form graph under construction.

| Schedule estimation | j60 | j90 | j120 | 30x10 | 30x25 | 60x10 |
|---|---|---|---|---|---|---|
| STA | $10^{4.0}$ | $10^{3.5}$ | $10^{3.1}$ | $10^{1.8}$ | $10^{1.3}$ | $10^{1.5}$ |
| SIM-1 | $10^{5.0}$ | $10^{4.6}$ | $10^{4.3}$ | $10^{3.1}$ | $10^{2.3}$ | $10^{2.6}$ |
| SIM-5 | $10^{4.4}$ | $10^{3.9}$ | $10^{3.6}$ | $10^{2.4}$ | $10^{1.6}$ | $10^{1.9}$ |
| SIM-10 | $10^{4.1}$ | $10^{3.7}$ | $10^{3.3}$ | $10^{2.1}$ | $10^{1.3}$ | $10^{1.6}$ |
| SIM-30 | $10^{3.6}$ | $10^{3.2}$ | $10^{2.9}$ | $10^{1.6}$ | $10^{0.8}$ | $10^{1.1}$ |
| SIM-60 | $10^{3.3}$ | $10^{2.9}$ | $10^{2.5}$ | $10^{1.3}$ | 3.0 | 7.6 |
| SIM-100 | $10^{3.1}$ | $10^{2.6}$ | $10^{2.3}$ | $10^{1.1}$ | 1.0 | 4.0 |

Table 6.4: Policy enumerations per minute (HEURISTICSEARCH).

completion of an activity $a_1$ as $\mu(C^{\Pi}_{a_1}) + \omega\sigma(C^{\Pi}_{a_1})$ where $\omega$ is a (positive) parameter than can be choosed arbitrarily. Suppose that the current image of the chain-form graph of a resource is as depicted in Figure 6.6 and that activity $a_2$ must be appended to 2 out of 3 chains. At the head of chains 1, 2, and 3 lie activities $a_1, a'_1$, and $a''_1$, respectively. Their stochastic completion times as estimated in the previous execution of step 3.a are also depicted.[4]

With $\omega = 0$ it will be appended to chains 1 and 2, adding edges $(a_1, a_2)$ and $(a'_1, a_2)$ to the policy under construction. However, with $\omega = 3$ it will be appended to chains 1 and 3. This decision is made in step 3.b.ii.A. of CFGCONSTRUCTION-CHAINING-HEURISTIC. Of course, empty chains, if existent, are given priority. To prioritize among chains, the heuristic relies on having estimated the stochastic completion times in step 3.a. That is, the stochastic completion times are re-estimated after each activity is put in the chain-form graphs of all resources (that it uses). It should be noted that the design of a competitive heuristic solver is not in the scope of this thesis. The design and use of this heuristic only serves the purpose of comparing the usefulness of LG-SSTA with that of simulation for taking such decisions. There is a great difference in performance in comparison to the RANDOMWALK procedure. For j120, on average, the objective function is about 40% smaller with $\omega = 0$ and about 42% smaller with $\omega = 3$. For 30x5, the objective function is about 60% smaller with $\omega = 0$ and about 62% smaller with $\omega = 3$.

All experimentation reported below involves the same groups of instances used for Experiment 1.b.

### 6.3.1 Experiment 2.a

Experiment 2.a is identical to Experiment 1.b, except now HEURSTICSEARCH is used instead of RANDOMWALK. Let us first see how estimating the stochastic schedule once after the chaining of each activity affects the rate of solution enumerations (Table 6.4). We observe that the enumeration rate drops by about two orders of magnitude. This is expected since the stochastic schedule is now estimated $O(|A|)$ times per solution instead of once.

[4] These completion times are not necessarily Gaussian but in line with LG-SSTA and for the sake of simplicity we shall make this assumption.

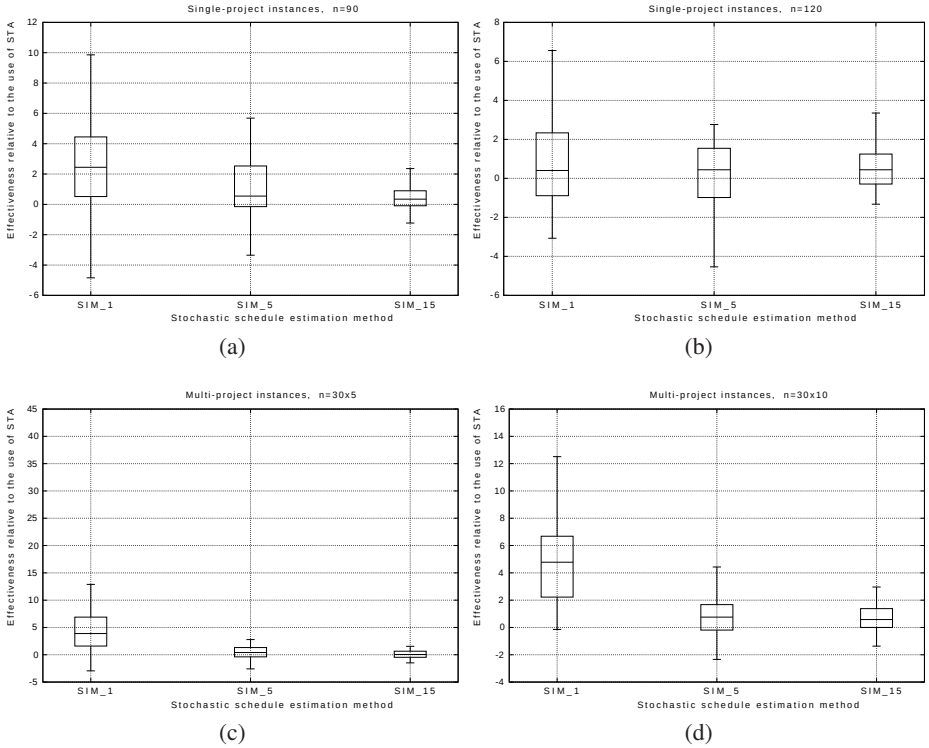


Figure 6.7: Effectiveness of HEURISTICSEARCH with $\omega = 0$, across different methods for evaluating the objective function.

This renders the heuristic approach for instances from the 30x25 and 60x10 groups impractical, especially when using more than 10 simulations or LG-SSTA.

However, an incremental update of the stochastic schedule can be performed after the chaining of an activity. Putting an activity to one or more chains corresponds to adding to it one or more immediate predecessors. Only the start/completion time of this activity and all its successors needs to be updated. Implementing LG-SSTA in an incremental fashion [71] we observe a speed-up of up to 550× for the instances in 30x25 and 60x10. The enumeration rate then becomes almost the same as before; i.e., when the stochastic schedule is only estimated once after all activities have been chained. An incremental re-estimation of the stochastic schedule is also possible with simulation but we have not implemented this functionality. For this reason, the following results have been obtained with non-icremental versions of LG-SSTA and simulation. (Note that we do not experiment with instances from 30x25 and 60x10–We only investigate the multi-project case, on instances from 30x5 and 30x10 just as in Experiment 1.b).

Figure 6.7 dthe same kind of information as Figure 6.4 except now HEURISTICSEARCH is used, with the heuristic parameter $\omega$ set to zero. We observe that LG-SSTA now dominates in most cases all other methods for providing information to the heuristic. A few simulations are not sufficient for maintaining the necessery level of accuracy, misleading

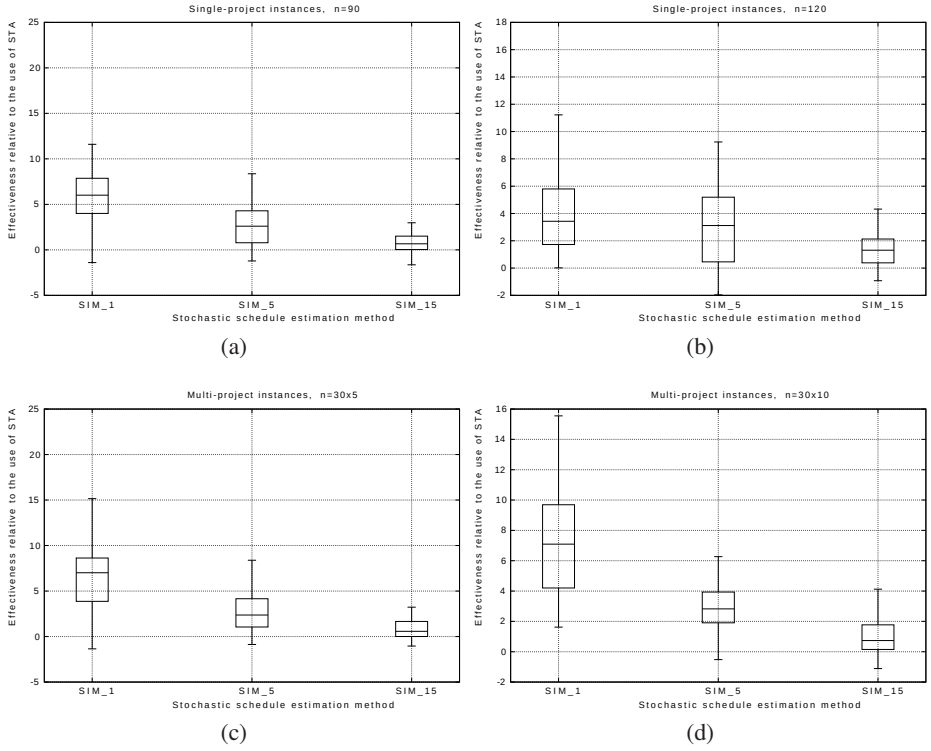


Figure 6.8: Effectiveness of HEURISTICSEARCH with $\omega = 3$, across different methods for evaluating the objective function.

the heuristic decisions. However, it is also apparent that informing the heuristic with 15 simulations is directly comparable to informing the heuristic with LG-SSTA since the difference in performance is very small.

### 6.3.2 Experiment 2.b

In the previous experiment we used $\omega = 0$. Thus, the variance of estimated timing quantities is not part of the information used in the heuristic. This experiment is identical to the previous one except now we let $\omega = 3$. This results in better performance throughout all groups of instances which makes the results of this experiment more important.

The new results are displayed in Figure 6.8. We know that a few simulations are usually not sufficient for a reliably estimating the mean of timing quantities in the stochastic schedule, and even more so for estimating the variance. By letting the variance also be part of the information used in the heuristic it can be seen that the performance gap between LG-SSTA and simulation widens.

### 6.3.3 Concluding remarks

Experiment 2 demonstrated the concept of estimating the stochastic schedule that corresponds to the current image of the solution under construction in order to guide the remaining steps. Experiment 1 suggests that a small number of simulations suffice for detecting an improvement in the best solution seen so far. For this reason, the accuracy of LG-SSTA appears to be an "overkill" when used for the same purpose. Experiment 2, however, suggests that this accuracy is, indeed, necessary when the estimated stochastic schedule is used in a heuristic mid-construction. In this case, LG-SSTA appears to be an efficient method able to provide the necessary level of accuracy.

# Chapter 7

# Conclusions and Future Work

The purpose of this thesis was to extend existing research on the scheduling process at NedTrain facilities. We focused on the particular challenge of mitigating the negative impact of uncertainty on the timely delivery of the maintenance projects. To enable a quantifiable representation of uncertainty, we proposed the Stochastic NedTrain Scheduling Problem (SNSP) model: a multi-project extension of the stochastic Resource-Constrained Project Scheduling Problem (SRCPSP). The main body of our work (Chapters 4 and 5) concentrated on how to deal with the shortcommings of present methods for finding robust ES policies.

We proposed efficient solution enumeration schemes by extending the works of Policella et al [56] and Artigues et al [3] to the area of stochastic scheduling. This enabled us to overcome the infeasibility of enumerating solutions by eliminating so-called Minimal Fodbidden Sets [65, 66] (for problem instances of practical size). Experiments show that a substantial enumeration rate can be maintained even on large multi-project instances with over 600 activities. In addition, we dealt with the performance bottleneck that arises from the use of crude Monte Carlo simulation, generally considered as the only practical means to evaluate the objective function in stochastic scheduling. The computational cost of simulation may be especially high when the objective function involves the estimation of variance to achieve the minimization of risk. Leveraging the PERT network-like nature of an ES policy, we proposed the use of Statistical Static Timing Analysis (SSTA) from the domain of robust VLSI circuit design. As a proof-of-concept, we described in detail the application of a fast SSTA method which supports durations with Gaussian variability and linear interdependencies. Experiments on instances with Gaussian variability in activity durations show that the computational cost of this SSTA method is roughly equal to that of 15 to 20 simulations.

In conclusion, the proposed enumeration schemes in combination with the proposed use of SSTA establish a framework for the design of competitive heuristic solver procedures. Our work is particularly oriented towards large-scale problem instances in which the durations of project activities exhibit severe variability.

## 7.1 Future Work

A natural extension of our research would involve the design of heuristics to guide enumeration towards policies of high robustness. As we explained in Chapter 4, there is a great deal of similarity between Partial Order Schedules and ES policies. One idea for future work is to generalize the the concept of flexibility used in Precedence Constraint Posting, to the area of stochastic scheduling. This would involve the use of information provided by an SSTA method during the construction of a policy. Effective Precedence Constraint Posting procedures (e.g., [56, 14, 74]) could then be extended for finding robust ES policy solutions to SRCPSP instances.

Another potential research direction would focus on an optimization objective that is particularly useful for NedTrain maintenance projects: that of minimizing the number of tardy projects (instead of overal tardiness). It is our expectation this particular optimization goal could benefit from heuristics that make use of accurate information in the stochastic schedule that corresponds to the policy under construction. The necessary level of accuracy could be provided with the use of SSTA at a low computational cost.

An additional research direction is that of extending the LG-SSTA method for the general class of pre-selective policies, with generalized AND/OR constraints. This research effort would involve implementing the statistical min operator.

Another future research direction would focus on the special nature of uncertainty in train maintenance projects. We remind the reader that the list of necessary repair operations on a train (maintenance project) remains unknown until inspection operations complete. A possible approach would be to associate each individual repair operation with a project activity. The stochastic duration $D_a$ of such an activity could be expressed as: $D_a = X$ with probability $p$, or $D_a = Y$ with probability $1 - p$. Stochastic variable $X$ describes the duration of the activity in case inspection shows that the repair is not necessary (with probability $p$). On the other hand, stochastic variable $Y$ describes the duration of the activity in case the repair is necessary.

Let us assume that $X$ and $Y$ are Gaussian. The distribution of $Y$ would have a mean at the origin and a variance that is tending to zero (i.e., approximating a Dirac delta function). The distribution of $D_a$ would be bi-modal in shape, with an impulse at the origin and a Gaussian peak around $\mu(Y)$. A possible research direction would be to extend the Linear-Gaussian SSTA method to support bi-modal duration distributions of this type. An idea would be to represent every timing quantity in the PERT network as a list of peaks and when two or more peaks are close enough they could be merged into a single peak. The research effort would then involve adjusting the statistical sum and max operators accordingly.

A different approach for handling the special nature of uncertainty in repair operations would involve representing the project plan by means of a Graphical Evaluation and Review Technique (GERT) network [57]. Beyond durational uncertainty, a GERT network also incorporates structural uncertainty; i.e., each activity is associated with a probability of occurrence. An additional research direction is to investigate the development of new classes of scheduling policies, specifically for problems in which the plan is represented as a GERT network. Yet another research direction would involve investigating the connection between a PERT network representation with activity durations that exhibit bi-modal

variability (as explained earlier) and the corresponding GERT network representation.

# Bibliography


[1] VG Adlakha and VG Kulkarni. A classified bibliography of research on stochastic pert networks: 1966-1987. *INFOR*, 27(3):272–296, 1989.

[2] Carlos Ansótegui, Miquel Bofill, Miquel Palahı, Josep Suy, and Mateu Villaret. Satisfiability modulo theories: An efficient approach for the resource-constrained project scheduling problem. *Resource*, 7(3):2, 2011.

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

[4] Christian Artigues, Sophie Demassey, and Emmanuel Neron. *Resource-constrained project scheduling*. Wiley-Iste, 2010.

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

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

[7] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer. Statistical timing analysis: From basic principles to state of the art. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(4):589 –607, april 2008.

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

[9] Peter Brucker, Andreas Drexl, Rolf Möhring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.

[10] Peter Brucker, Sigrid Knust, Arno Schoo, and Olaf Thiele. A branch and bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 107(2):272–288, 1998.

[11] John M Burt and Mark B Garman. Conditional monte carlo: A simulation technique for stochastic network analysis. *Management Science*, 18(3):207–217, 1971.

[12] JA Carruthers and Albert Battersby. Advances in critical path methods. *OR*, pages 359–380, 1966.

[13] A. Cesta, A. Oddi, and S.F. Smith. Profile based algorithms to solve multiple capacitated metric scheduling problems. In *Proceedings of the*, volume 4, pages 214–223, 1998.

[14] A. Cesta, A. Oddi, and S.F. Smith. Iterative flattening: A scalable method for solving multi-capacity scheduling problems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 742–747. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2000.

[15] Hongliang Chang and Sachin S. Sapatnekar. Statistical timing analysis considering spatial correlations using a single pert-like traversal. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, ICCAD '03, pages 621–, Washington, DC, USA, 2003. IEEE Computer Society.

[16] Charles E. Clark. The greatest of a finite set of random variables. *Operations Research*, 9(2):145–162, 1961.

[17] Richard L. Daniels and Janice E. Carrillo. β-robust scheduling for single-machine systems with uncertain processing times. *IIE Transactions*, 29:977–985, 1997.

[18] N. Dawood. Estimating project and activity duration: a risk management approach using network analysis. *Construction Management & Economics*, 16(1):41–48, 1998.

[19] F. Deblaere, E. Demeulemeester, W. Herroelen, and S. Van de Vonder. Robust resource allocation decisions in resource-constrained projects*. *Decision Sciences*, 38(1):5–37, 2007.

[20] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1):61–95, 1991.

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

[22] Erik L Demeulemeester and Willy S Herroelen. *Project Scheduling: A Research Handbook*, volume 102. Kluwer Academic, 2002.

[23] Bajis Dodin. Approximating the distribution functions in stochastic networks. *Computers & operations research*, 12(3):251–264, 1985.

[24] R. Evers, 2009. Algorithms for Scheduling of Train Maintenance, Delft University of Technology, Master's Thesis.

[25] A.A. Fernandez, R.L. Armacost, and J.J.A. Pet-Edwards. The role of the nonanticipativity constraint in commercial software for stochastic project scheduling. *Computers & Industrial Engineering*, 31(1):233–236, 1996.

[26] Donald L Fisher, Donna Saisi, and William M Goldstein. Stochastic pert networks: Op diagrams, critical paths and the project completion time. *Computers & operations research*, 12(5):471–482, 1985.

[27] Cristiano Forzan and Davide Pandini. Statistical static timing analysis: A survey. *Integration, the VLSI Journal*, 42(3):409 – 435, 2009.

[28] D. Golenko-Ginzburg and A. Gonik. Stochastic network project scheduling with non-consumable limited resources. *International Journal of Production Economics*, 48(1):29–37, 1997.

[29] J.F. Gonalves, J.J.M. Mendes, and M.G.C. Resende. A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research*, 189(3):1171 – 1190, 2008.

[30] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[31] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5(2):287–326, 1979.

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

[33] Sonke Hartmann and Rainer Kolisch. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 127(2):394–407, 2000.

[34] Andrei Horbach. A boolean satisfiability approach to the resource-constrained project scheduling problem. *Annals of Operations Research*, 181(1):89–107, 2010.

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

[36] J. Jaffari and M. Anis. On efficient monte carlo-based statistical static timing analysis of digital circuits. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 196–203. IEEE Press, 2008.

[37] J.A.G. Jess, K. Kalafala, S.R. Naidu, R.H.J.M. Otten, and C. Visweswariah. Statistical timing for parametric yield prediction of digital integrated circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(11):2376–2392, 2006.

[38] George B Kleindorfer. Bounding distributions for a stochastic acyclic network. *Operations Research*, 19(7):1586–1601, 1971.

[39] R. Kolisch, C. Schwindt, A. Sprecher, et al. Benchmark instances for project scheduling problems. *INTERNATIONAL SERIES IN OPERATIONS RESEARCH AND MANAGEMENT SCIENCE*, pages 197–212, 1999.

[40] Rainer Kolisch. Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management*, 14(3):179–192, 1996.

[41] Oumar Kone, Christian Artigues, Pierre Lopez, and Marcel Mongeau. Event-based milp models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3–13, 2011.

[42] V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI magazine*, 13(1):32, 1992.

[43] P. Leach. Modeling uncertainty in project scheduling. In *Proceedings of the 2005 Crystal Ball User Conference*, 2005.

[44] R. Leus. Resource allocation by means of project networks: dominance results. *Networks*, 58(1):50–58, 2010.

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

[46] Arfst Ludwig, Rolf H Möhring, and Frederik Stork. A computational study on bounding the makespan distribution in stochastic project networks. *Annals of Operations Research*, 102(1):49–64, 2001.

[47] J. H. Roseboom C. E. Clark W. Fazar Malcolm, D. G. Application of a technique for research and development program evaluation. *Operations Research*, 7(5):646–669, 1959.

[48] J. J. Martin. Distribution of the time through a directed, acyclic network. *Operations Research*, 7(5):46–66, 1965.

[49] R. H. Mohring, F. J. Radermacher, and G. Weiss. Stochastic scheduling problems i–general strategies. *Mathematical Methods of Operations Research*, 28:193–260, 1984. 10.1007/BF01919323.

[50] R. H. Mohring, F. J. Radermacher, and G. Weiss. Stochastic scheduling problems ii–set strategies. *Mathematical Methods of Operations Research*, 29:65–104, 1985. 10.1007/BF01918198.

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

[52] Frank T Mooty. *An introduction to PERT cost: Technical documentary report no. ESD-TDR-64-208*. LG Hanscom Field, 1964.

[53] Emmanuel Neron. Lower bounds for the multi-skill project scheduling problem. In *Proceeding of the Eighth International Workshop on Project Management and Scheduling*, pages 274–277. Citeseer, 2002.

[54] D. O'Connor. Exact and approximate distributions of stochastic pert networks. *Dublin, University College*, 2006.

[55] N. Policella, A. Oddi, S. Smith, and A. Cesta. Generating robust partial order schedules. *Principles and Practice of Constraint Programming–CP 2004*, pages 496–511, 2004.

[56] N. Policella, S.F. Smith, A. Cesta, and A. Oddi. Generating robust schedules through temporal flexibility. In *Proceedings of the 14th international conference on automated planning & scheduling, ICAPS04*, pages 209–218, 2004.

[57] A.A.B. Pritsker. Gert-graphical evaluation and review technique. 1966.

[58] Franz Joseph Radermacher. Scheduling of project networks. *Annals of Operations Research*, 4(1):227–252, 1985.

[59] Pierre Robillard and Michel Trahan. The completion time of pert networks. *Operations Research*, 25(1):15–29, 1977.

[60] Craig W Schmidt and Ignacio E Grossmann. The exact overall time distribution of a project with uncertain task durations. *European Journal of Operational Research*, 126(3):614–636, 2000.

[61] D Sculli. The completion time of pert networks. *Journal of the Operational Research Society*, pages 155–158, 1983.

[62] C Elliott Sigal, A Alan B Pritsker, and James J Solberg. The stochastic shortest route problem. *Operations Research*, 28(5):1122–1129, 1980.

[63] S.F. Smith, M.A. Becker, et al. An ontology for constructing scheduling systems. In *Working Notes of 1997 AAAI Symposium on Ontological Engineering*, pages 120–127, 1997.

[64] S.F. Smith and C.C. Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of the National Conference on Artificial Intelligence*, pages 139–139. JOHN WILEY & SONS LTD, 1993.

[65] F. Stork. *Stochastic resource-constrained project scheduling*. PhD thesis, Berlin Technical University, 2001.

[66] F. Stork and M. Uetz. On the generation of circuits and minimal forbidden sets. *Mathematical programming*, 102(1):185–203, 2005.

[67] S. Tasiran and A. Demir. Smart monte carlo for yield estimation. In *Proc. ACM/IEEE TAU*, 2006.

[68] M. Uetz. *Algorithms for deterministic and stochastic scheduling*. Cuvillier, 2001.

[69] Richard M Van Slyke. Letter to the editormonte carlo methods and the pert problem. *Operations Research*, 11(5):839–860, 1963.

[70] V. Veetil, D. Sylvester, and D. Blaauw. Efficient monte carlo based incremental statistical timing analysis. In *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pages 676–681. IEEE, 2008.

[71] Ravindran K. Kalafala K. Walker S.G. Narayan S. Beece D.K. Piaget J. Venkateswaran N. Hemmett J.G Visweswariah, C. First-order incremental block-based statistical timing analysis. In *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pages 2170 – 2180. IEEE Council on Electronic Design Automation, Oct. 2006.

[72] Jerome D Wiest. A heuristic model for scheduling large projects with limited resources. *Management Science*, 13(6):B–359, 1967.

[73] M. Wilson, N. Roos, B. Huisman, C. Witteveen, P. De Causmaecker, J. Maervoet, T. Messelis, K. Verbeeck, and T. Vermeulen. Efficient workplan management in maintenance tasks. In *Proceedings of the 23rd Benelux Conference on Artificial Intelligence*, pages 344–351. CODeS/KaHo Sint-Lieven, 2011.

[74] M. Wilson, C. Witteveen, B. Huisman, M. Wilson, N. Roos, B. Huisman, C. Witteveen, P. De Causmaecker, J. Maervoet, T. Messelis, et al. Enhancing predictability of schedules by task grouping. In *Proceedings of the 20th European Conference on Artificial Intelligence*, pages 344–351. IOS Press, 2012.

[75] Ningxiong Xu, Sally A McKee, Linda K Nozick, and Ruke Ufomata. Augmenting priority rule heuristics with justification and rollout to solve the resource-constrained project scheduling problem. *Computers & Operations Research*, 35(10):3284–3297, 2008.

[76] Hong-Quan Xue, Sheng-Min Wei, and Yang-En Wang. Resource-constrained multi-project scheduling based on ant colony neural network. In *Apperceiving Computing and Intelligence Analysis (ICACIA), 2010 International Conference on*, pages 179 – 182, dec. 2010.

# Appendix A

# Example policies

We hereby display two policies $\Pi$ and $\Pi'$ that have been found as solutions to SNSP problems $J = (P, Z, D, q)$ and $J' = (P, Z, D, q')$, respectively. Both instances share parameter $P = (A, E)$ depicted in Figure A.1. This plan consists of two projects, P1 and P2, which in turn consist of activities 1 to 33 and 34 to 66, respectively. The sink of P1 is activity 33 and the sink of P2 is activity 66. Furthermore, both problems have common resource constraints $Z$ and stochastic activity durations $D$. It should be noted that P1 and P2 have the same structure. However, the activities in P2 have, on average, longer durations than those in P1.

Problems $J$ and $J'$ have different due-date vectors. In $J$ the due-date of P1 is about four times that of P2 (i.e., $q_{33} = 4q_{66}$). Policy $\Pi = (A, E \cup H)$ has been found as a solution to $J$ and is displayed in Figure A.2. Clearly, tardiness is minimized by giving priority to the activities of the project with the strict due-date, namely P2. For instance, a close inspection shows that P1 practically does not start (specifically, activities 4 and 5) until activities 49, 50, 52, and 61 of P2 have completed. In practice, this means that one or more resources (e.g., one or more crews of engineers) must finish work in the train that corresponds to P2 before they move to work on the train that corresponds to P1.

Conversely, in $J'$ the due-date of P2 is about four times that of P1 (i.e., $q'_{66} = 4q'_{33}$). Policy $\Pi' = (A, E \cup H)$ has been found as a solution to $J'$ and is displayed in Figure A.3. Now tardiness is minimized by giving priority the activities of P1.

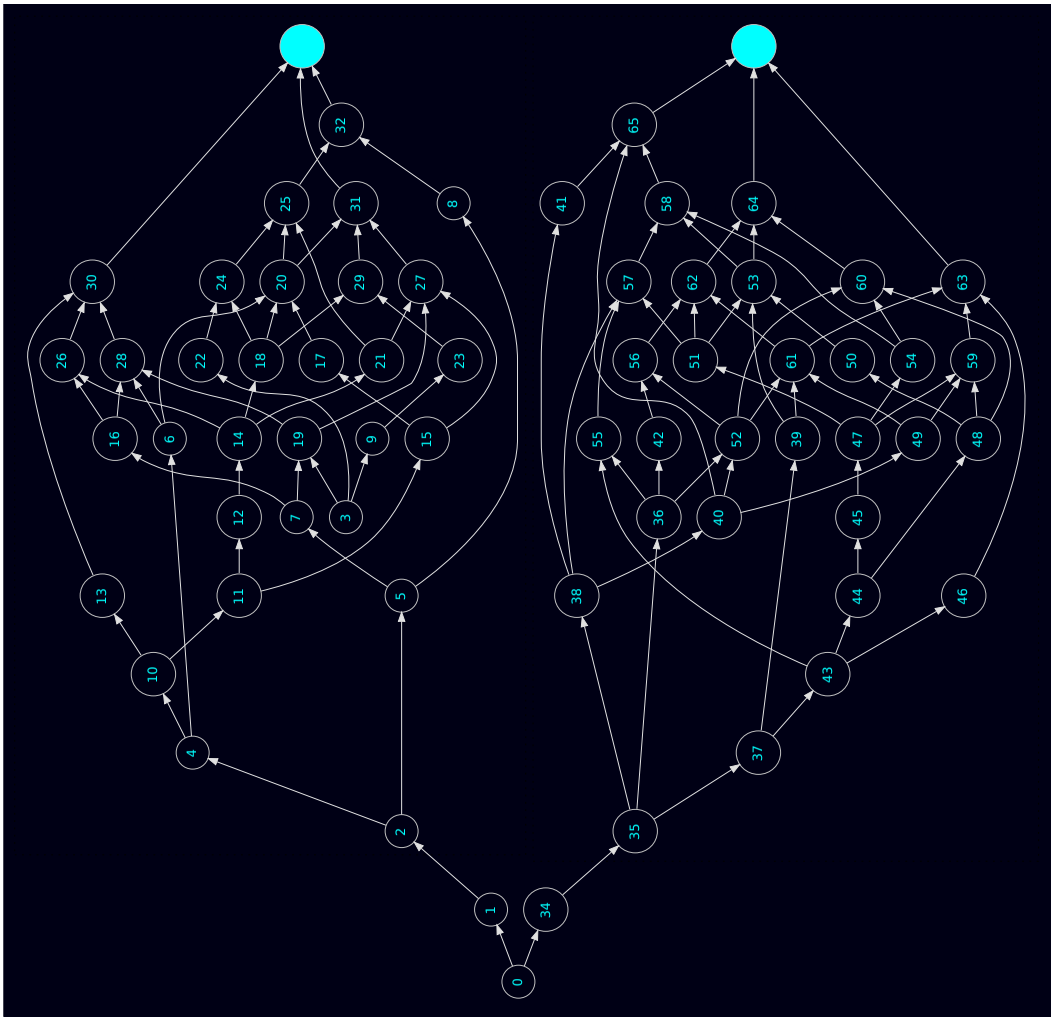


Figure A.1: Plan $P = (A, E)$ with two projects (rotate by +90 degrees). Colored nodes 33 and 66 correspond to the sinks of projects P1 and P2 respectively.


Figure A.2: Solution $\Pi = (A, E \cup H)$ to problem $J$. Colored edges eliminate forbidden sets in $P$. Clearly, priority is given to the activities of project P2.


Figure A.3: Solution $\Pi' = (A, E \cup H')$ to problem $J'$. Colored edges eliminate forbidden sets in $P$. Clearly, priority is given to the activities of project P1.