Optimization Methods for Real Life Scheduling Problems

Larsen, Rune; Bang-Jensen, Jørgen

Publication date:
2012

Document Version
Publisher's PDF, also known as Version of record

Optimization Methods for Real Life Scheduling Problems

PHD THESIS

prepared at:

University of Southern Denmark

Department of Mathematics and Computer Science

By

Rune Larsen

July 9, 2012

Thesis Advisor: Jørgen Bang-Jensen

Abstract

In this thesis the challenges that arise when applying optimization methods to real life scheduling problems are considered. The work is divided into four main parts:

i) Real life problems posed by two companies have been investigated first hand. The primary focus of this work in the context of the thesis is identifying and handling the constraints exogenous to the core optimization problem. Each of the problems can be formulated as a *variable size bin packing problem*, and contains a "subset sum" sub problem, that is solved by dynamic programming. Complications include strict real time requirements and uncertain information resulting in a stochastic problem. These are handled by regular reoptimizations by heuristics with the "anytime" property. The optimization methods proposed have been adopted, and are currently being employed in a production environment in both companies.

ii) The subject of the ROADEF/EURO Challenge 2010 is considered. The problem is based on the French electricity production which is primarily done by nuclear power plants. The problem is stochastic due to uncertainty in estimated demand and varying amounts of electricity from other sources such as hydro power plants. This inherent stochasticity is represented as scenarios for demands. The goal is to produce a maintenance schedule and a production plan. The problem contains a large set of constraints, that makes even finding feasible solutions challenging. The proposed solution consists of a multi phased approach, applying constraint programming to obtain an initial solution, which is subsequently improved by local search heuristics applied to a surrogate problem. Finally the solution to the surrogate problem is generalised to a solution to the actual problem. After correcting a programming error, the obtained results of the solution method comes out ahead of all competitors.

iii) A highly modular framework for dynamic rescheduling for problems with solutions representable as temporal networks is proposed. The framework provides a method for using a deterministic solver to generate an initial solution to a stochastic instance, implementing and executing strategies for rescheduling based on Monte Carlo sampling, and visualise stochastic solutions. The framework is demonstrated on stochastic versions of two different scheduling problems from the literature: The job shop instance FT10 and a "single machine weighted completion time" problem. For each problem, different rescheduling policies are tested and evaluated. The most significant predictor of solution quality of the problems under consideration, is shown to be the policies for generating the deterministic instances for rescheduling.

iv) The framework from part *iii)* is used to generate initial solutions to a train scheduling problem from the Utrecht area in the Dutch railway system. Three algorithms from the literature are considered and the sensitivity of solutions produced by these algorithms to delays (of trains) is analysed. Delays are taken from a uniform distribution for travel times and Weibull distributions for dwell times. The policy for generating deterministic instances to solve, is shown to be a better predictor for quality of the stochastic solution, than any other parameter including the type of solver used.

Finally a set of directions for future work with the framework is proposed, including two planned papers on applications of the developed framework: Applying a closed loop rescheduling strategy to the train scheduling problem, and applying the framework on a real life anodizing plant problem.

Summary (In Danish)

Denne afhandling omhandler de udfordringer der opstår når man anvender optimerings algoritmer på skedulerings problemer fra den virkelige verden. Den er delt op i fire dele:

i) To konkrete optimerings problemer stillet af to firmaer er blevet behandlet. Det primære fokus i denne kontekst var, at identificere og agere på de betingelser der lå ud over det grundlæggende optimeringsproblem. Begge problemer kunne formuleres som "variable size bin packing" problemer og indeholdt et "subset sum" delproblem, der blev løst med dynamisk programmering. Processen kompliceres af skarpe tids rammer for alogoritmens kÃ¸retid og stokastisk information der giver anledning til et stokastisk problem. Disse bliver håndteret ved regelmæssige genoptimeringer af "anytime" heuristikker. De foreslåede optimeringsmetoder er i brug i begge virksomheder i et produktions miljø.

ii) I denne del behandles emnet for ROADEF/EURO Challenge 2010. Problemet er baseret på elektricitets produktionen i Frankrig, der primært drives af nukleare kraftværker. Problemet er stokastisk da det reelle strømforbrug samt produktion fra alternative energikilder er ukendt. Stokasticiteten er repræsenteret ved scenarier for efterspørgsel af energi. Målet er at lave en vedligeholdelsesplan for udskiftning af brændstof samt en produktions plan. Problemet indeholder en stor mængde forskellige betingelser der skal opfyldes. Dette gør det svært bare at finde en lovlig vedligeholdelses plan. Den præsenterede løsning består af en algoritme der arbejder i tre faser; en lovlig løsning bliver fundet med constraint programming, løsningen bliver forbedret med lokalsøgning på et surrogat problem. Til sidst generaliseres løsningen på surrogatproblemet til det reelle problem.

Efter at have rettet en fejl i algoritmen outperformede den alle konkurrenter.

iii) I denne del præsenteres et framework til dynamisk reskedulering af problemer hvis løsninger kan repræsenteres som et temporalt netværk. Frameworket giver mulighed for at bruge en deterministisk algoritme til at skabe og visualisere en løsning til det stokastiske problem og implementere reskedulerings strategier baseret på Monte Carlo sampling. Frameworket demonstreres på stokastiske versioner af to forskellige skedulerings problemer fra litteraturen: Job shop instansen FT10 og et "single machine weighted completion time" problem. For hvert problem testes og evalueres forskellige reschedulerings strategier. Den vigtigste parameter i denne proces viser sig at være strategien for at generere deterministiske instanser til den anvendte algoritme.

iv) Frameworket fra *iii)* bliver anvendt til at generere initielle løsninger til et togskedulerings problem fra Utrecht området i det hollandske togsystem. Tre algo-

ritmer fra litteraturen anvendes og deres løsningers følsomhed overfor forstyrrelser i togplanen evalueres. Forsinkelserne er taget fra uniform fordelinger for togenes rejsetid og fra Weibull distributioner for togenes stop. Strategien for at generere deterministiske instanser til algoritmerne bliver vist at have større betydning for løsningens kvalitet end den anvendte algoritme. Dog bevarer den bedste deterministiske algoritme sit forspring selv efter stokasticiteten er tilføjet til problemet.

Til slut præsenteres retninger for fremtidigt arbejde med frameworket. Dette inkluderer blandt andet to planlagte videnskabelige artikler: Anvendelse af frameworket til closed loop reskedulering på togskedulerings problemet samt anvendelse af frameworket på et anodiserings anlæg fra industrien.

Acknowledgments

Several persons have been contributed to the development of this thesis. First mention goes to my advisor Jørgen Bang-Jensen without whom no thesis would have been made.

Chapter 2 would not have been possible without the cooperation and support from the Anonymous company and Danfoam, both companies have contributed with real life instances, feedback and encouragement.

I participated in the ROADEF/EURO Challenge 2010 with Steffen Godskesen, Thomas Sejr Jensen and Niels Kjeldsen, and the competition proved an amazing experience. Not least because of the attitude of my team mates. We would also like to thank Marco Chiarandini whose encouragement and extremely thorough proofreading was invaluable.

Midway through my work, I got in contact with Marco Pranzo. The original idea was to cooperate on project, but it turned out to set the direction for the rest of the work in my thesis. His encouragement and continued support cannot be overestimated.

The work on robustness relies heavily on data and solvers obtained through cooperation with Francesco Corman, Andrea D'Ariano, Dario Pacciarelli and Marco Pranzo. Collaborating with them on the robustness paper has been a pleasure.

Keeping sane and productive requires good company and good work environment, both of which IMADA has provided. There are more people who deserve an explicit mention than there is room for, but you know who you are.

My family have also been supportive through the whole process, and they certainly deserve the acknowledgement.

Last but by no means least, I would like to thank Maja Frederiksen for her patience and continued support.

Odense, March 2012, Rune Larsen

Contents

Chapter 1

Introduction

Contents

1.1 Introduction

1.1.1 Motivation

Scheduling problems are among the most researched problems in the literature, and they have many real-life applications. In order to take advantage of these applications several barriers have to be overcome:

- Making companies aware of what science can provide, and at what cost.
- Cheaply and quickly assessing the magnitude of possible improvements to justify investments.
- Academic problems must be reformulated to take special constraints into consideration.
- Any stochasticity must be taken in account.
- Algorithms must be made ready for a production environment.

Parts of these points are potentially of interest to operations research, and have been investigated to various degrees. To address all of these, one would need a framework that allowed for fast prototyping without making unnecessary assumptions about constraints or degrees of stochasticity, and which provided feedback of a form interpretable for laymen given a proper presentation.

1.1.2 Overview and Contribution of Ph.D. Thesis

To investigate the practicalities involved in solving real-life instances of combinatorial optimization problems, three different real-life problems are initially considered. The experience obtained in these chapters led to the conclusion, that generality of the optimization method, and the ability to handle stochasticity are essential. A framework is then introduced, that facilitates analysis of and addressing the stochasticity in optimization problems with solutions representable as a temporal network. Finally the framework is used to analyse a real-life instance of the train scheduling problem.

Chapter 2: A Real-Life Variable Size Bin Packing Problem explores two real-life cases of optimization problems. The aim of the cooperation with the two companies, was to investigate the practical issues arising in applying methods from the literature to a real life problem.

The main contributions are:

- A real life case of the variable size bin packing problem is presented, and the factors leading to difficulties in applying methods from the literature are outlined.
- Two algorithms are currently being used in a production environment, and the feedback has been overwhelmingly positive. This includes suggestions for new projects.
- An exact method has been presented for small variable size bin packing problems.
- A destruct/reconstruct heuristic for the variable size bin packing problem is presented, and shown to outperform state of the art methods consistently on a class of instances.[1]

Chapter 3: Scheduling Outages for Nuclear Power Plants explores a real-life problem posed as the ROADEF challenge 2010. This problem differs significantly from the ones in Chapter 2, as it contains a lot of diverse constraints, a challenging scheduling component and works with large instances. The problem also contains a significant stochastic component represented as scenarios, and they must be taken into account in the planning phase.

The main contributions are:

- A three-phase hybrid heuristic for scheduling of nuclear power plants is presented. The problem was posed in the ROADEF/EURO Challenge 2010. The current version of the heuristic would have ranked first in the competition, but an implementation bug meant that we finished second in the junior category and sixth[2] overall.

[1] This class contains the real life instances described.
[2] Seventh if multi threaded solvers are included.

- The problem under consideration is shown to be NP-hard.
- The proposed heuristic uses a range of precomputed data structures and in particular approximations in order to reduce the running times of the algorithms.

Chapter 4: A Framework for Dynamic Rescheduling A framework for dynamic rescheduling is introduced, and demonstrated on a "single machine weighted completion time problem" and a classical job shop problem. The framework is highly modular and supports swapping solvers, objective functions, scenarios, probability functions and solutions during execution. It also provides visualizations for decision makers to base their decisions upon.

The main contributions are:

- A highly flexible framework for dynamic rescheduling.
- Strategies for generating deterministic instances, increasing robustness of the generated solutions for the two problems under consideration.

Chapter 5: Robustness of Train Schedules A real life train scheduling from [Corman 2011b] is investigated using the framework presented in chapter 4. The initial hypothesis was that solutions computed by more the advanced Branch and Bound (B&B) method might be a lot more sensitive to disruptions than the solutions computed by the myopic FIFO. This hypothesis was rejected as the gap in solution quality grew nearly as often as fell when disruptions were introduced.

The main contributions are:

- The framework for dynamic rescheduling is shown to be able to handle large instances.
- B&B from [Corman 2011b] was shown to produce more robust solutions than FIFO.
- A set of possible applications of the framework is presented.

Chapter 6: Conclusions sums up the results obtained in the thesis, and outlines directions for future work.

The work in Chapter 2 has been published as [Bang-Jensen 2012] and the work from Chapter 3 has been submitted as [Godskesen 2011], received a positive first review and been resubmitted. Chapter 3 has also been published as a popular science article, and Chapter 2 was presented at the EURO XXIV LISBON conference.

The work in Chapter 4 and 5 has been documented in two drafts ([Larsen 2012]) and will be submitted to a scientific journal. The work from Chapter 5 has also been presented at the APMOD conference 2012 by Francesco Corman.

1.2 List of Papers and Articles

The first entry is the authors master thesis that inspired some of the work. The rest of the work has been done during the PhD study.

Palletflow and Sampling Sequences in Connection with Retrieval of Goods from a Warehouse (Master thesis)
Rune Larsen
January 2008

Efficient Algorithms for Real-Life Instances of the Variable Size Bin Packing Problem
Jørgen Bang-Jensen and Rune Larsen
Accepted in Computers & Operations Research 2012

Solving a real-life large-scale energy management problem
Steffen Godskesen, Thomas Sejr Jensen, Niels Kjeldsen and Rune Larsen
Submitted to Journal of Scheduling

ROADEF/EURO Challenge 2010 - Skedulering af atomkraft i energisektoren (Popular science article)
Steffen Godskesen, Thomas Sejr Jensen, Niels Kjeldsen and Rune Larsen
ORbit 2010

A Framework for Dynamic Rescheduling
Rune Larsen and Marco Pranzo
Being submitted early April 2012

Robustness of Optimal Train Schedules to Stochastic Disturbances of Process Times
Andrea D'Ariano, Francesco Corman, Rune Larsen, Dario Pacciarelli and Marco Pranzo
Extended abstract accepted at the APMOD conference 2012

Robustness of Optimal Train Schedules to Stochastic Disturbances of Process Times
Andrea D'Ariano, Francesco Corman, Rune Larsen, Dario Pacciarelli and Marco Pranzo
To be submitted

1.3 Notation, Concepts and Definitions

Notation and definitions specific to each chapter will be introduced therein, and as the problems under consideration differ significantly and use a lot of notation each, variable names may be defined differently in different chapters. Some of the notation pertains to multiple chapters however and will be introduced here.

1.3.1 Challenges in Real Life Problems

There are multiple ways of defining what a real life problem is in academia. The obvious definition would be a problem that is present outside of academia, but even the most theoretical simplified problem might have such an application. A real life problem will in this thesis be defined as a problem with a real life application showing characteristics that are not considered standard in academia. Note that a lot of research has been applied trying to solve such problems, resulting in a progressing frontier. ([Schöbel 2006, Pinedo 2009, Ouelhadj 2009] and more)

1.3.1.1 Non-Standard Constraints

When approaching a company, you often find no exact match for their problems in the literature. The problem is usually recognisable as a version of bin packing, timetabling, vehicle routing, job shop scheduling or a similarly well researched problem, but additional constraints are typically enforced and the objective function(s) might be different.

Optimization algorithms can be divided into *exact algorithms*, *approximation algorithms* and *heuristics*, all of which might be suitable for the problem under consideration. Exact (or approximation if no exact is available) algorithms are often preferred in academia if they exists and are sufficiently fast, but in real life applications preferences can change for multiple reasons: *i)* The constraints that differ from the academic problem might not be representable in the exact and approximation approaches. (e.g. nonlinear constraints) *ii)* The extra constraints might prolong the running time of the algorithm unacceptably. *iii)* The running time of a solver is often constrained, and having even a small chance of producing no solution might be unacceptable. *iv)* Companies often fail to communicate their problem in its entirety,[3] and at any point in time a constraint, causing one of the previously mentioned problems to occur, could be added. For these reasons, heuristics are often considered though there exists exceptions such as one of the problems described in Chapter 2.

From point *iv)* it is clear that as few assumptions as possible must be made about the absence of constraints. Sometimes models and methods from the literature can be chosen for their increased expressiveness rather than their strict performance on the problem under consideration.

[3] Or the problem changes at a later point in time.

1.3.1.2 Stochasticity

A common complication encountered in real life problems is that information is not necessarily known ahead of time, leading to a *stochastic problem*. For some stochastic problems statistics are well known and available, leading to *probability functions* describing the distribution for each uncertain element. If no such statistics exists, the probability functions must be estimated or bounded by the company to make optimization feasible.

Handling stochasticity in the literature is done either *proactively* or *reactively*. In the proactive approach, solutions are constructed so that they are *robust* to *disruptions* of the solution, while the reactive approach attempts to adapt the solution online to incoming disruptions. Note that a reactive approach does not exclude a proactive one, and that some disruptions such as breakdown of trains cannot be handled proactively. In some applications an initial solution is created in the *planning phase* trying to take the stochasticity into account proactively, whereafter a reactive approach is applied to minimise the impact of disruptions.

The proactive approaches alone are generally considered insufficient for real life problems as described in [Ouelhadj 2009] where a detailed review of the current 'state of the art' in reactive approaches can be found.

To evaluate the effect of stochasticity a simulation approach is often used. Simulation is done by generating a set of scenarios by sampling the probability functions, and evaluating the solutions on these. The purpose of the evaluation can be both ensuring feasibility and increasing average quality.

When dealing with a stochastic problem, *ex-ante* denotes that only the information available before any disruption has occurred, is available when solving, and *ex-post* indicate all information is available. The optimal ex-post solution thus represents a theoretical lower bound on any solution obtained by rescheduling.

1.3.2 Multiple Objectives and Pareto Optimality

Real life problems often contain more than a single objective to be optimized. Sometimes (as in the Danfoam case in Chapter 2) the objectives are ranked, and can thus often be treated independently in order of significance. If the objectives cannot be ranked according to significance, they must be considered simultaneously.

A solution is *Pareto optimal* if no solution exists that is better with respect to all objectives simultaneously. The set of all Pareto optimal solutions is called a *Pareto frontier* and one such is shown in figure 1.1. Solutions for with there exists another solution improving all objectives is *dominated*.

There are multiple strategies used in the literature for handling multiple different objectives. All of them involves an *decision maker* as there is no mathematical way of combining different objectives without an implied subjective decision. The decision maker can make the decisions:

A priori: The decision maker is involved before the optimization algorithm is run, and the involvement usually takes the form of ranks as in chapter 2, weights

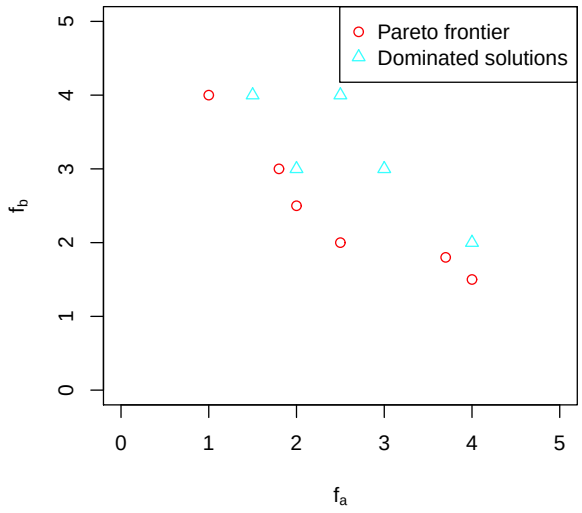

Figure 1.1: A Pareto frontier for an imaginary multi objective (f_a, f_b) minimization problem. Red circles are Pareto optimal while the rest of the solutions represented is dominated by one or more Pareto optimal solutions.

as in chapter 3 or an added constraint on the solution quality wrt. one of the objectives, whereafter that objective is ignored but kept as a constraint.

The purpose of the a priori approach is to create a single objective, that can be evaluated by the applied optimization algorithm.

A posteriori: The decision maker is involved after the optimization, where a set of solutions are presented for the decision maker to choose from. The optimization algorithm will often attempt to generate a subset of the Pareto optimal solutions for this purpose. For some problems explicit diversification might be needed.

Interactively: The decision maker is involved in the actual search process, typically by being given choices at regular intervals such as: What objective should be optimized next? What objective is allowed to deteriorate? Which of a set of solutions should be explored further? What should be rescheduled? When should a rescheduling occur? This approach allows adaptation to some concepts that cannot be included in the model such a some concepts of fairness,[4] but requires frequent attention from the decision maker.

[4]In many practical applications fairness can be described mathematically, but for a real life case experienced by the author, timetables were rejected in favour of "more fair" solutions, even when the predefined fairness was perfectly balanced.

A detailed survey can be found in [Ehrgott 2000], and the Ph.D thesis [Jensen 2011] treats the issue in more detail.

1.3.3 Scheduling

We define scheduling similarly to ([Hoos 2004]): A set of jobs $\mathcal{J} := \{J_1, \ldots, J_n\}$ each consist of a set of *operations* $J_i := \{o_{i1}, \ldots, o_{im(i)}\}$. The operations have to be processed on a machine from the set $\mathcal{M} := \{M_1, \ldots, M_m\}$. A *schedule* is a mapping of jobs to machines and processing times subject to *feasibility constraints* and *optimization objectives*. For some problems, notation for enumerating the operations without specifying the job associated is beneficial, and in these cases we refer to an operation as o_j (a single subscript).

Scheduling problems can be defined by a symbolic notation such as $1||\Sigma w_j C_j$ and $J10|r_j|C_{max}$. The notation consists of three parts: *i)* the machine environment and the number of machines. *ii)* the feasibility constraints and *iii)* the objective function. $1||\Sigma w_j C_j$ thus corresponds to a single machine weighted completion time problem, while $J10|r_j|C_{max}$ denotes a 10 machine job shop problem with release times with the objective of minimizing the makespan. I will assume this notation is known, but if needed definitions can be found in [Hoos 2004] page 422.

1.3.3.1 Temporal Networks

A solution to a scheduling problem can often be represented as a *temporal network* ([Roy 1964, Elmaghraby 1995, Mascis 2002]) implemented as a graph $G = (N, F \cup S)$ where N is a set of nodes representing the operations and $F \cup S$ is a set of edges representing precedence relations. A *precedence relation* (i, j) is a constraint on the starting time of operation o_j which must be greater or equal to the starting time of the predecessor o_i plus a deterministic given *delay/processing time* p_{ij}. The number p_{ij} which is known in advance and can assume positive, zero or negative values.

Precedence relations are divided into two sets: *fixed* F and *selected* S. Fixed precedence relations are present in all solutions to the specified problem, while selected precedence relations depend on scheduling decisions.

In the temporal network there are two dummy nodes representing the source and the sink. These nodes respectively precedes and succeeds every other node. Without loss of generality we assume that the source operation start at time 0. A longest path from the source to any other node will determine the corresponding operations earliest possible start time. Positive length cycles denote an infeasible solution.

The primary weakness of raw temporal networks is its inability to express resource constraints, but this issue has been treated by [Laborie 2003, Bidot 2009].

1.3.3.2 The Alternative Graph Model

The alternative graph model ([Mascis 2002]) and its predecessor the disjunctive graph model, work by representing the problem as a temporal network $G = (N, F \cup$

A), where A is a set of *alternative* precedence relations.

Alternative precedence relations are partitioned into pairs, i.e., $((i,j),(h,k)) \in A$. A *schedule* is thus an assignment of starting times $t_0, t_1, \ldots, t_n$ to operations $o_0, o_1, \ldots, o_n$ respectively, such that all fixed precedence relations, and exactly one for each pair of the alternative precedence relations, are satisfied. Given a pair $((i,j),(h,k)) \in A$ in a solution we have either $(i,j) \in S$ or $(h,k) \in S$, where S is the set of selected arcs. Scheduling decisions are in form of disjunctions. That is, one of the two alternative disjunctive arcs have to be selected, i.e., added to the current graph.

The alternative graph model is known to be able to model constraints such as blocking, release times, due dates, synchronization of operations, "no wait" constraints and more. [Larsen 2008, Mascis 2002]

Chapter 2

A Real-Life Variable Size Bin Packing Problem

Contents

2.1 Contribution of the Author

This chapter is an extended version of the paper [Bang-Jensen 2012] which was written in collaboration with my supervisor Jørgen Bang-Jensen.

The development of the heuristics presented, getting the algorithms ready for a production environment and writing the initial draft for the paper is the work of the author.

Working with companies on real-life applications creates a significant overhead in work that is not publishable by itself. This cannot be avoided as real life instances are necessary for understanding process of adapting to real life applications. The reward is when, as here, the companies adopt the proposed algorithms and use them in their production.

2.2 Introduction

Several real-life problems can be formulated as packing or cutting problems or relaxed versions thereof. Therefore packing and cutting problems, have been studied extensively, and since Gilmore and Gomery published their integer programming based approach [Gilmore 1961], most methods have been based on this idea. The method is based on solving an integer programming problem with a reduced set of columns each corresponding to some packing patterns of a single bin, and generating additional columns as necessary.

This method does not perform as well on bin packing problems where the bin size might vary, as increasing the number of bin sizes, increases the number of columns that must be added to the basis set before an optimal solution of the LP relaxation is reached. An overview of the literature on the variable size bin packing problem (VSBPP) can be found in [Correia 2008] and [Wäscher 2007], and most employed methods are based on Gilmore and Gomerys work.

The common approaches suffer from multiple potential drawbacks when considering real-life instances coming from industrial companies:

- Some companies cannot guarantee a single bin size, but require the size to vary from bin to bin, thus increasing the number of packing patterns dramatically. For example, this situation can arise due to deformation of the material during or after a casting process.

- Some companies require that the code can be run on embedded systems with very limited memory and sometimes even without online memory allocation, thus excluding the use of LP solvers.

- Some companies gain more knowledge about the problem data online, and require this information to be taken into account immediately. They might also require a solution within milliseconds while allowing a possible negative impact on its quality.

- Some companies are capable of reusing unused capacity in the last bin packed. This situation often arises when modelling a cutting problem, where cutting can be resumed on the last used item, given that it is large enough to warrant storage.

Motivated by these difficulties, our aim was to develop and implement a construction heuristic and a local search heuristic, which could produce high quality solutions very fast, when a large number of bin sizes are available and at the same time the algorithms should adapt fast to online changes in data.

2.2.1 Two Real Life Problems

Our work is motivated by the following real-life problems which we have encountered through collaboration with industrial companies.

2.2.1.1 An Anonymous Company

The first company produces machinery that uses a combination scale to create packages of certain weights based on a number of smaller compartments containing material of known weights. At any given point in time, they can ask for a certain weight of material to be released, and the machine should open the combination of compartments creating the closest possible weight that exceeds the requested amount. After this the compartments just emptied are refilled and their new weight is determined.

A further complication arises when some compartments are not filled before the next release of material. When this can be predicted, we wish to ensure that the compartments that are released, leave a set of filled compartments with the best possible options for new compartments to release. This corresponds to calculating a packing of two bins, where the wasted space in the first is considered more important than that of the second. This problem is relatively easy except for the fact that in one of the real-life applications we consider it is extremely time-critical, and can never take in excess of 100 milliseconds even on an embedded processor.

The first problem is easily solvable, using an optimized version of the dynamic programming heuristic described in [Kellerer 2004] which is usually employed to solve subset sum problems. While the second problem basically corresponds to a bin packing problem.

2.2.1.2 Danfoam

The company Danfoam A/S produces high quality foam mattresses for a world wide market. The first part of the production involves cutting prescribed length foam blocks (typically from 110 to 230 cm) from larger blocks. These larger blocks which reside in a storage are either previously unused blocks whose lengths are around 30 m, or they are remainders from a previous cutting process in which case the length lies between 8 m and 29 m.

The exact data (available to us and used in some of the tests) cannot be described here due to non-disclosure agreements, but the overall structure of the daily cutting problem is as follows; a substantial number (several hundreds) of items whose lengths are in the interval between 110 cm, and 230 cm (about 10 different lengths) are to be cut from blocks residing in the storage. These block lengths will vary due to setting after the casting process, and possibly previous cuts taken of that block. The amount of waste should be minimized with the additional condition that the last block used may be returned to storage if it is at least a certain length, say 8 m (in which case no waste is counted for this block).

A further complication arising when optimizing the cutting pattern is that, due to the properties of the foam material, the length of a block might change slightly as it is brought in for cutting. Furthermore some items might be damaged during production and have to be rescheduled on the remaining blocks to be cut.

Thus overall we are faced with a variable sized bin packing problem with a large number of different bin sizes which are typically much larger than the item sizes (usually 10-15 items or more fit in a random bin) and where there are relatively few item sizes (about 2-10 different in the range 110 cm to 230 cm). Due to the online changes in the data, the time requirements are strict.

2.3 Results From the Literature

The literature concerning cutting and packing is split due to the difference in characteristics of the problems considered, inferred by small variations in the input data. Classification schemes were proposed in [Dyckhoff 1990] naming the problem 1/V/D/R or 1/V/D/M, but the scheme was found lacking and expanded upon in [Wäscher 2007] naming the problem a Residual Bin Packing Problem or Residual Cutting Stock Problem. This demonstrates a major problem when classifying real life problem types and solvers efficiency: A real life problem might well contain instances that are of widely different types, and a solver might be usable on a wide array[1] of different types of problems.

The following characteristics are significant when considering a problem:[2]

- Input minimization vs. output maximization.
- Objective function strongly correlated/equal to wasted residual objects vs. objective function weakly correlated or independent from amount of wasted residual objects.
- Homogeneous objects vs. heterogeneous objects.
- Homogeneous items vs. heterogeneous items.

[1] e.g. cutting stock problems and variable sized bin packing problems.

[2] Objects are the bins or rolls, and items are the items or cuts in the packing and cutting terminology respectively.

- Low average number of items per object vs. a high number of items per object in a good solution.

Besides this, real life problems usually contain side constraints such as:

- No waste can be reused later and thus ignored vs. a set number of items with waste above a certain length can be ignored vs. all waste above a certain length can be ignored. These concepts are often called usable leftover in the literature such as [Cui 2010] [Cherri 2009] etc.

- Short time available for solving vs. long time available for solving.

When considering real life problems one should specify what range the above characteristics can occur in, and likewise when talking about a solution method, it should be stated which ranges of the above characteristics it can be brought to work on, and how it might influence performance.

The Danfoam problem is an input minimization problem with heterogeneous objects and instances with either heterogeneous or homogeneous items. It has a relatively high number of average items per object (10-15), one can ignore waste in *one* of the bins if the waste is large enough, and instances must be solved in a very short time frame.

The exact heuristics have grown increasingly promising for similar problems as seen in [Belov 2002], [Alves 2008] and most recently [Haouari 2009b], but as cuts and bounds become more advanced, factoring in the possible waste on a single bin and similar real-life constraints, become more intractable.

Recent work on heuristics for the variable size bin packing problem [Haouari 2009a] has shown some impressive results. Problems are solved extremely fast, and often to optimality. However the problems considered in this chapter do have a few vital differences from the ones considered in [Haouari 2009a] and other articles.

- Bins can contain a larger number of items, massively increasing the number of columns a column generation based solver would have to create. A genetic algorithm would have to increase population size significantly, to make sure that all possible assignments of items to bins can be created by crossovers.

- Bins are very unlikely to share size with other bins, creating problems similar to the mentioned above.

- Any algorithm is *required* to be able to deliver an answer anytime after a very short initialization time, thus preventing a lot of column generation approaches from producing solutions in time. This includes the demand, that the algorithm must be able to resume optimization fast, given any changes in either bin or item sizes.

- The number of item sizes are limited, which often requires a change of assignments for a large number of items, before a better solution can be reached.

- In one of our applications, waste concentrated on the last bin can be safely ignored, as long as it remains above a certain size.

Other recent work [Koch 2009] [Dimitriadis 2009] has been concerned with real life problems where leftover was considered usable, but they allow waste on all bins, thus enabling the integration of this side constraint into a column generation scheme. The same problem is encountered when considering the paper [Cherri 2009]. The paper [Cui 2010] does consider the case where the number of usable leftovers is bounded, but it allows only a closed set of leftover lengths, and bounds on the number of each of these separately to integrate them into an integer program.

2.4 Notation and Problem Definition

The bin packing problem considered is as follows:

Given k bins $B_1 \dots B_k$ of sizes $v_1 \dots v_k$ respectively, and n items x_1, $x_2 \dots x_n$ of sizes $a_1 \dots a_n$ respectively: Find a partition of items $x_1 \dots x_n$ into sets $S_1, S_2, \dots, S_k \in S$ such that $\sum_{x_i \in S_j} a_i \leq v_j$ for $j = 1 \dots k$.

Two variants of the problem are considered in this chapter, and the objective function depends on the variant under consideration.

2.4.1 Classical Bin Packing

In the classical bin packing problem (BPP), all bins are of the same size v, thus $v_1 = v_2 = \dots = v_k = v$. The wasted space, in any solution to this problem, depends only on the number of bins utilized. Thus the objective function normally just focuses on minimizing this number. Papers such as [Falkenauer 1992] have employed more advanced objective functions, to distinguish between the attractiveness of solutions utilizing the same number of bins.

When using an algorithm that optimizes with respect to a different objective function f_a that does not match the actual objective function f, some properties can be helpful to maintain: given two solutions A, B in a minimization problem.

- $f(A) < f(B) \iff f_a(A) < f_a(B)$ preserving a strict preference for better solutions.
- If $f(A) = f(B)$ and $f_a(A) < f_a(B)$ then A should contain characteristics that are considered preferable or likely to lead to an improving solution.
- Evaluation of the contribution of each bin to f_a should be possible, thus allowing for Δ-evaluations.

In this chapter, the quality of a solution is considered to be the sum of the waste in the solution, and ties are broken favouring the solution having the largest sum when summing up the waste per bin squared. This creates a situation where solutions with an equal amount of waste, are considered better the more the waste is concentrated. Experiments have verified this strategy to improve the local search.

2.4.2 A Generalization Allowing Varying Bin Sizes

In many real life instances such as the ones treated in this chapter, the assumption that the bin size is constant does not hold. In that case $v_1, v_2, \ldots, v_k$ might differ, and the number of bins used fail to express the space wasted by an assignment of items to bins. In these cases the objective is often to reduce the sum of the unused capacity in the used bins.

Minimize $\sum_{S_j \in S} w_j$

where $w_j = \begin{cases} v_j - \sum_{x_i \in S_j} a_i & \text{if } |S_j| > 0 \\ 0 & \text{otherwise} \end{cases}$

For some real life instances, that arise from cutting stock like problems such as the one mentioned from Danfoam, the last bin packed can contain spare room, that is not considered waste, as it can be used in the next packing. This is not trivial to model when using the standard column generation based techniques, but as we will show later, it is achievable using the developed local search heuristic.

2.5 Subproblems

The bin packing problem can be seen as several interdependent subset sum problems. Several efficient methods, as described in [Kellerer 2004], [Pisinger 1999] and [Pferschy 1999], exist for solving these given sufficiently small bins or items. As this is typically the case for bin packing problems, these methods can be used for exploration of neighbourhoods and for construction heuristics, by providing optimal packings of a single bin.

2.5.1 A Single Bin

The simplest subproblem consists of assigning items to a single bin, without concern for subsequent assignments. This can be achieved by using dynamic programming to compute all possible sums of item sizes. The sum closest to but not higher than the bin size is now chosen, and the items giving the sum are then found by backtracking in the dynamic programming table. This corresponds to construction heuristics used in [Haouari 2009a], [Fleszar 2002] and analyzed in depth in [Caprara 2004] and [Caprara 2005]. Our implementation differs from theirs, allowing for subsequent use in an exact solver.

Table 2.1 demonstrates the concept. The first column signifies that the sums 0 and 3 can be made with the item seen so far. The second column that the sums $\{0, 2, 3, 5\}$ can be made with the two items seen so far.[3]

A sum can be achieved by combining a subset of the items seen so far, if and only if the number in the corresponding row and the last column is 1. The sum can be achieved *without* the item at the head of the column if and only if the previous column has a one in the same row. Likewise the sum can be achieved *by using*

[3] The size of the item is shown above each column.

Sum \ Item size	3	2	4	...	1
0	1	1	1	...	1
1	0	0	0	...	1
2	0	1	1	...	1
3	1	1	1	...	1
4	0	0	1	...	1
5	0	1	1	...	1
⋮	⋮	⋮	⋮	⋱	⋮
s	0	0	0	...	?

Table 2.1: Dynamic programming table. s is the maximum sum that is computed, which is usually the size of the bin to fill.

the item corresponding to the column if and only if the preceding column has a 1 in the row x places above, where x is the size of the item corresponding to the column. Using this method recursively as shown in Algorithm 1 on page 20, all combinations of items yielding a given sum can be enumerated. Its worth noting that an exponential number of these might exist.

2.5.1.1 Favoring Use of Large Items

Larger items often makes fitting items into bins harder.[4] To favour combinations using these large items the following strategy has been employed:

Before construction of the dynamic programming table, the items are sorted according to size in increasing order. During backtracking, items are thus encountered in decreasing order, and the algorithm simply chooses an item, if it can be part of the desired sum. Similar preferential treatment for larger items has been suggested in [Haouari 2009a], [Fleszar 2002] and [Zhang 2000] amongst others.

2.5.1.2 Avoiding Duplicate Subsets

A common problem encountered when working with sets of numbers is that the number of representations of the same set can explode when multiple items have the same size.

Lets say we have five items with the size a. If a set of items fills out a bin, and contains exactly one item of size a, we know that four more such sets exist that use the other items of size a. From a bin packing point of view, these sets are not different from the one we have already seen, and we would like to generate only one of these sets. This is done by sorting items before the dynamic programming table is created, and enforcing that once an item of a certain size is skipped during backtracking, all subsequent items with the same size must also be skipped. This

[4] This is the motivation for applying the first fit *decreasing* heuristic.

ensures that if k items of size a is chosen, it will always be the first k in the ordering thus eliminating generation of equivalent solutions.

2.5.1.3 Optimizations

It should be noted that the dynamic programming table conceptually consists of boolean values, and that the operations computing a new column are the same for all its boolean entries. Namely $b_{i,j} = b_{i-1,j} \vee b_{i-1,j-x}$ where i indexes columns, j indexes rows, and x is the size of the item corresponding to the i'th column.[5]

The columns in the table can thus be represented by sets of integers corresponding to the value of each column interpreted as a bitstring, and all operations can be performed as bitwise boolean operations between the integers. This yields a significantly improved though asymptotical equivalent running time in a critical region of the code. In our practical applications this optimization is of great importance.

2.5.2 An Exact Solver for VSBPP

As the bin packing problem is NP-hard, no polynomial time algorithm can be expected to solve the problem to optimality. For sufficiently small instances however, there might exist algorithms that finish quickly enough to be viable. The definition of 'quickly enough' varies from application to application. In some of the interactive applications, two seconds might be very acceptable, while a quarter of a second is pushing the limit when interacting with fast paced machinery. Recent work in the field of exact solvers includes the following articles mentioned in the literature review: [Haouari 2009b] uses branch and bound to solve instances with an unlimited supply of each bin size and both convex and concave cost functions. [Belov 2002] and [Alves 2008] solves a multiple length cutting stock problem that is equivalent to the problem treated in this chapter aside from the low number of cutting stock lengths, high number of item sizes, their relation between item and bin sizes and their inability to handle usable leftovers. If problems that have these characteristics are considered, the local search proposed can easily be adapted to use these as solvers instead.

Our exact solver (Algorithm 2 and 3) uses the dynamic programming approach to solve the subset sum subproblems, it then iterates through the solutions attempting to solve the induced problem recursively. To make the approach faster, several optimizations have been made:

- The iteration is performed in an order that ensures that once a feasible solution is found, it is optimal and the search can be halted.
- Several fail fast checks have been implemented, detecting that no solutions exists before all possible assignments have been enumerated.

[5] Out of bound values are assumed to be false.

Data: A dynamic programming table $dpt_{sum,i}$.
Item sizes $a_1 \dots a_n$ corresponding to each column as described in section 2.5.1 sorted in increasing order as described in section 2.5.1.1.
A maximum index of usable item m.
The last used item size l.
A desired sum s.
Result: A list of items that sums to s, or false if this is impossible.

if $s = 0$ **then** `// We are done as we are trying to sum to 0`
  return $\emptyset$;
end
if $a_m \geq s$ **then** `// The item is too large to use`
  return *readSolution*($dpt_{sum,i}$, $a_1 \dots a_n$, $m-1$, l, s);
end
if $m = 0 \vee dpt_{s,m} = 0$ **then** `// No solution with the desired sum`
  return *False*;
end

`/* Can create a solution using the m'th item? */`
if $dpt_{s-a_m,m-1} = 1 \wedge l \neq a_m$ **then**
  result $\leftarrow$ readSolution($dpt_{sum,i}$, $a_1 \dots a_n$, $m-1$, a_m, $s-a_m$);
  if *result* $\neq$ *False* **then**
    return $a_m \cup$ *result*;
  end
end
`/* Can create a solution not using the m'th item? */`
if $dpt_{s,m-1} = 1$ **then**
  result $\leftarrow$ readSolution($dpt_{sum,i}$, $a_1 \dots a_n$, $m-1$, l, s);
  if *result* $\neq$ *False* **then**
    return $a_m \cup$ *result*;
  end
end
`/* Only reachable if similar sized items were skipped. */`
return *False*;

Algorithm 1: readSolution($dpt_{sum,i}$,$a_1 \dots a_n$,m,l,s) The next solution can be obtained by blocking the use of the item a_i with the lowest i in the previous solution, and unblocking all a_j for $j < i$

- If there exists an upper bound b on the waste that the optimal solution can contain. The number of bins of certain size can be removed as long as enough bins of that size remain to contain all items plus b units of waste.

Data: A set $Bins$ of k bin sizes $v_1 \dots v_k$
A set $Items$ of n item sizes $a_1 \dots a_n$
A minimal amount of waste that can be ignored in a bin ($minIgnored$)
Result: An optimal partition of items into bins

Compute all possible sums of bin sizes $v_1 \dots v_k$;
Compute all possible sums of item sizes $a_1 \dots a_n$;
$itemSum \leftarrow \sum_{i=1}^{n} a_i$;
$maxSpill \leftarrow max(0, max(v_1 \dots v_k) - minIgnored)$;
for $waste \leftarrow 0$ **to** $infinity$ **do**
  for $spill \leftarrow 0$ **to** $maxSpill$ **do**
    if ($waste + itemSum - spill$) *is an obtainable sum of bin sizes and* ($spill$) *is an obtainable sum of item sizes* **then**
      for *Each binSet with sum of sizes* ($waste + itemSum - spill$) **do**
        if $max\{v_i | v_i \in Bins \setminus binSet\} - spill > minIgnored$ **then**
          continue;
        end
        for *Each itemSet with sum of sizes* ($itemSum - spill$) **do**
          $solution \leftarrow$ attemptAssign($binSet$,$itemSet$,$waste$);
          if $solution \neq null$ **then**
            assign ($Items \setminus itemSum$) to an unused bin in $solution$;
            return $solution$;
          end
        end
      end
    end
  end
end

Algorithm 2: The algorithm searches for solutions with an increasing amount of waste. For each waste amount it searches through solutions assigning increasing amounts of items ($spill$) to the bin that has its waste ignored.

2.5.2.1 Iteration Order

To ensure that the first feasible solution found is also optimal, a variable keeps track of the amount of wasted space allowed. This variable is initialized to 0. If the current bin cannot be filled causing less than that amount of wasted space, there is no solution with that amount of waste, and the subproblem is infeasible. If

Data: A set $Bins$ of k bin sizes $v_1 \dots v_k$
A set $Items$ of n item sizes $a_1 \dots a_n$
An allowed amount of waste in the bins: $slack$
Result: An optimal partition of items into bins

if $k = 0$ **then** `// If we have filled all bins`
  return *Empty solution*;
end
$itemSums \leftarrow$ Compute all possible sums of item sizes $a_1 \dots a_n$;
if *failFastChecks(*$Bins, slack, itemSums$*)* **then**
  return *null*;
end
$surplus \leftarrow 0$;
while $surplus < v_1$ *and* $surplus < slack$ **do**
  for *Each set itemSet of items summing to* $v_1 - surplus$ **do**
    $solution \leftarrow$ attemptAssign($bins \setminus v_1$,$Items \setminus itemSet$,$slack - surplus$);
    if $solution \neq null$ **then**
      return $solution \cup itemSetassignedtov_1$;
    end
  end
  $surplus \leftarrow surplus + 1$;
end
return *null*;

Algorithm 3: attemptAssign($Bins$,$Items$,$slack$) This algorithm assumes that all bins need to be used. The function failFastChecks is described in Section 2.5.2.2

the initial problem is deemed infeasible, the waste variable is incremented, and the search repeated.

In the traditional case of a uniform bin size, a further optimization can be realized by ensuring that the waste plus the sum of the items always correspond to a multiple of the bin size.

2.5.2.2 Fail Fast Checks

The first fail fast optimization consists of checking whether the sum of item sizes plus the allowed waste, equals something obtainable by combining bins. For traditional bin packing problem this corresponds to checking if it equals a multiplum of the bin size, whereas it can be achieved by using the dynamic programming approach on the bin sizes if they are allowed to vary.

The second fail fast check is done each time items are assigned to a bin. If the minimum number of bins needed for the remaining items cannot be filled without incurring more than the allotted wasted space, the partial assignment can lead to no feasible solution. This check is made by relaxing the remaining problem to allow the use of all items in all bins, and can be done trivially using the dynamic programming table.

2.6 Solution Methods for the BPP

As the goal was to create a method that did not rely on external solvers, and gave solutions superior to those obtainable with First Fit Decreasing (FFD) etc., local search was considered. Firstly an initial solution had to be constructed, and in addition to First Fit Decreasing and Best Fit, a dynamic programming approach as seen in [Caprara 2004] and [Caprara 2005] was considered as construction heuristic.

2.6.1 A Dynamic Programming Construction Heuristic

To generate initial solutions for the local search heuristic, a dynamic programming approach is used. First it generates all possible sums of item sizes, and greedily assigns the best fitting set of items to the first bin. Then it solves the remaining problem recursively.

For the problem faced by the undisclosed company, this construction heuristic produces guaranteed optimal combinations of compartments to release when considering a release event locally. When using the dynamic programming table, it's trivial to iterate through all combinations of compartments satisfying this. The ability to use the remainder of the compartments efficiently given a set of compartments to release can be measured by rerunning this construction heuristic on the remaining compartments. Should this technique approach the time limit, the best found solution so far can always be returned instantly.

2.6.2 A Local Search Heuristic

The local search heuristic (DPLS) (as outlined in Algorithm 4) takes an initial solution generated by the dynamic programming construction heuristic. It then creates subproblems of a more manageable size, and tries to reoptimize them to optimality.

Other initial solution generation schemes can be chosen, but the algorithm performs better given a solution where as many bins as possible are packed very well, as opposed to having every bin packed reasonably well. Experiments showed that using FFD or similar construction heuristics for generating an initial solution decreased solution quality and/or increasing computation time on average.

Given a feasible solution to the problem, the heuristic first checks if there is wasted enough space to potentially use one bin less. If this is the case, the heuristic identifies the bins containing wasted space and chooses a subset of these again ensuring that they contain enough wasted space to potentially free one bin. Besides these sets, one or more full bins might also be chosen to increase the size of the neighbourhood.

Given the subset X of bins found as above, the heuristic undoes all assignments of items to members of X, and attempts to reassign the items to the available bins[6] optimally using the exact solver. A new solution is accepted if it uses fewer bins, or if it concentrates the waste more.[7] The concentrated waste criteria was added to bring the local search out of local minimas, without requiring the release of a very large number of bins.

2.7 Generalizing the Method to Our Real-Life Application

In the real life problem of Danfoam, the bin sizes will vary and the optimization objective is to minimize the wasted bin space in the used bins, possibly ignoring waste in the last bin if it exceeds a certain amount. This requires adaptations in both the construction heuristics and the local search heuristic.

2.7.1 Dynamic Programming Construction Heuristic for VSBPP

The approach can easily be generalized to the case where bins vary in sizes: Instead of assigning to the first bin, the potentially wasted space is computed for each bin, and the one with the least amount of potentially wasted space is chosen. To break ties, the smaller or larger of the bins can be chosen, depending on what seems preferable in the problem.[8]

[6] That is, X and all previously unused bins

[7] If the sum of the squared waste in each bin is higher.

[8] Smaller bins are usually chosen, as they usually contain fewer items, and the possible sizes obtainable with fewer items tend to be fewer.

```
Data: An initial solution sol.
nbReleased ← 2;
while The allotted time has not expired and the solution is not verifiably
optimal do
    X ← selectNextSubset(nbReleased);
    subS ← reassign items from X optimally;
    if cost(subS) < cost(X) then
        Reassign items from X in sol according to their assignment in subS;
    end
    if All releasable subsets have been explored since last improvement then
        nbReleased ← nbReleased + 1;
    end
end
return sol
```

Algorithm 4: A general local search based on dynamic programming. The differences between the local search algorithms used in this chapter consists in 1) the ways the subsets X are selected. 2) The check for whether optimality is achieved and 3) in the objective function used.

2.7.2 Dynamic Programming Based Local Search Heuristic for VS-BPP

Like in the case with constant bin sizes, a set of bins are selected and all their assignments are released. The first motivator for change consists in the fact, that a better solution might be reached, even if the sum of the wasted bin space does not exceed the size of a bin. This can be seen by considering a case where a bin has wasted space, and a smaller bin exists that can contain the same items.

The second difference is that bins that are in use, and whose wasted space exceeds the size of the smallest released element can have its wasted space considered as a bin in its own right in the subproblem. This further reduces the need to release assignments on bins with wasted space.

To exploit these properties, relatively few bins are released initially, and the number is increased until the time allotted is spent. To demonstrate the consequences of various time limits, graphs of cost vs. time are presented in Figures 2.1 and 2.2 on page 31.

When adapting the objective function to the alternative one, where a bin with more waste than 800 cm can be considered having no waste, it is important to note that our exact solver will iterate through candidate solutions in a way that seeks to concentrate waste on the last bin considered. Thus when a subproblem is integrated into the current solution, it tends to have the waste concentrated, and the local search algorithm (DPLSW) can just use the new objective function as acceptance criteria for any candidate solution.

2.8 Performance and Comparisons

Measuring the performance of the developed heuristic is not trivial, as each set of benchmark instances are tailored towards a specific configuration of the problem. The instances used in [Falkenauer 1992] can be found at the OR library [Beasley 2009], and they provide a reasonable estimate of performance on instances with the same bin to item size ratio even though they are traditional bin packing problems. The instances used in [Haouari 2009a] differ significantly from the ones encountered in the real-life problems under consideration, in that it focuses on problems with very few items per bin, and a larger number of item sizes while still making waste in a large number of bins necessary. The instances from [Poldi 2009] have multiple bin sizes, and relatively few item sizes, but their relative size only matches the encountered real life problem on a few of the classes of problems defined.

Generators and instances used in this chapter can be obtained by contacting the authors.[9]

Coding was done in Java 1.6 and in all tests, a single thread was used on a Core i7 920 processor having 4 gigabytes of ram, and unless otherwise noted, an upper bound of ten seconds time usage has been enforced on each instance. Given enough time, the algorithm can usually improve the solution, and given enough time and memory it will give the optimal solution, but for all but the smallest instances from [Poldi 2009] this has not been done. On the real life instances, graphs are presented to show the quality of the solutions as a function of the time used.

2.8.1 Dynamic Programming Construction Heuristic

The obvious benchmark for a construction heuristic must be a test against the current most popular approach, which until lately was the first fit decreasing algorithm as seen in [Zhang 2000] and [Poldi 2009] amongst others.[10] The average[11] number of bins used can be seen in Table 2.2, and can be compared to the best solution known. This shows that the dynamic programming algorithm outperforms FFD on average and gets very close to optimality. On the instances from [Falkenauer 1992], DPC and FFD find the best solution in 133 and 2 of the 140 instances respectively, and tie on the remaining 5, thus we must conclude that DPC nearly always yields better solutions than FFD. This price of this improvement is that DPC does not run in polynomial time in the size of the bins, but this seems less alarming as the time it takes for both algorithms to finish all instances combined is measured in a few seconds. Furthermore it should be noted that the instances have a higher bin size than typically encountered as described in [Applegate 2002], and that the real life applications of DPC has proven its efficiency on much larger instances as seen in subsection 2.8.3.

[9] Note that the data from Danfoam is not freely available.

[10] In [Haouari 2009a] and [Fleszar 2002] they also propose subset sum, dynamic programming based solutions, and the method is rapidly gaining acceptance.

[11] The number of bins used, averaged over all instances.

The DPC matches the currently best known solution on 28 of the instances, and often this can be proven to be an optimal solution as relaxing the constraint that items cannot be partially assigned, gives the same cost in number of bins used.

Current best	DPC	FFD	DPLS
234.4833	235.15	237.4667	234.6667

Table 2.2: DPC VS. FFD average waste over all 140 instances. DPLS is described in subsection 2.8.2.

A generation scheme for test instances for general one-dimensional cutting stock problems can be found in [Poldi 2009], and Table 2.3 clearly shows that DPC outperforms the other presented constructive heuristics.

Class	K	m	v_1	v_2	best constr.	DPC	best resi.
1	3	5	0.01	0.2	118.75	250.25	108.70
2	3	5	0.01	0.8	42429.35	1739.48	683.05
3	3	5	0.1	0.8	31493.55	2309.60	617.20
4	3	20	0.01	0.2	170.50	274.90	155.65
5	3	20	0.01	0.8	49565.05	5349.08	392.60
6	3	20	0.1	0.8	84775.60	6016.45	361.40
7	5	10	0.01	0.2	152.30	303.38	129.20
8	5	10	0.01	0.8	73920.20	3821.23	545.00
9	5	10	0.1	0.8	96475.80	3780.48	785.10
10	5	20	0.01	0.2	163.80	293.65	147.60
11	5	20	0.01	0.8	81946.20	5943.08	172.05
12	5	20	0.1	0.8	98766.15	7733.68	192.90
13	7	10	0.01	0.2	146.05	380.38	115.90
14	7	10	0.01	0.8	87379.30	3342.33	241.35
15	7	10	0.1	0.8	118647.90	4018.05	247.65
16	7	20	0.01	0.2	190.30	511.78	126.00
17	7	20	0.01	0.8	91638.20	6751.58	132.60
18	7	20	0.1	0.8	125865.15	6754.70	153.65

Table 2.3: Testruns on instances from [Poldi 2009], K is the number of bin sizes, m is the number of item lengths, and v_1 and v_2 are the upper and lower bounds on item sizes relative to average bin size. Bin sizes are chosen uniformly randomly between 100 and 1000. The best results of the constructive and the residual algorithms presented in [Poldi 2009] can be seen in the columns best constr. and best resi. respectively.

2.8.2 Dynamic Programming Based Local Search

As the performance of solvers is highly dependent on the characteristics of the problem, finding a fair basis for comparisons can be challenging. The public test instances having the most similar bin and item size ratio with the encountered real life problem from Danfoam, are the instances in [Beasley 2009]. They are however single size bin packing problems, and all results obtained should be treated accordingly. The proposed local search heuristic does however manage to improve on the results produced by the dynamic programming construction heuristic on the instances from [Falkenauer 1992], creating results that are only about 0.18 bins or 0,078% above the currently best known solution as seen in table 2.2.

Table 2.3 and 2.4 contains results based on 400 instances per class generated as described in [Poldi 2009][12] They have multiple though fairly few "bin" sizes, but they suffer from a smaller item to bin size ratio on many of the classes. On the instances with the highest bin to item size ratio, the proposed local search consistently outperforms the heuristics proposed in [Poldi 2009] within a quarter of a second, and on many of the remaining instances the local search delivers better or comparable results fast, as can be seen in Table 2.4. The running time in [Poldi 2009] was an *average* of 2.27 – 2.59 seconds on a Pentium III (866 MHz - 256 MB RAM) depending on the heuristic. As the Core i7 920 processor and its 4 GB RAM is significantly more powerful, a *maximum* running time of 0.25 seconds in Table 2.4 is the fairest column to compare to, but the ability of DPLS to make use of extra time should be taken into consideration.

The actual 40 instances per class of [Poldi 2009] were unobtainable, and the 400 instances used in this chapter was thus generated independently, which this further complicates direct comparisons. DPLS outperforms all of the three proposed heuristics in [Poldi 2009] consistently on the six classes where $v_2 = 0.2$, and the likelihood of that happening under the assumption that DPLS is no better than the heuristics in [Poldi 2009] is bounded upwards by $0.5^6 = 1.5625\%$. Based on this and the rest of Table 2.4 we conclude that DPLS is better than the heuristics proposed in [Poldi 2009] on these instances and observe that it is competitive on most of the remaining instances.

On some of the instances with $v_2 = 0.8$ and $m = 20$, the local search performs less impressively, and a closer inspection revealed this to be due to two separate factors. DPLS performs better when there exists a solution with very little waste, as the search for the optimal solution will terminate faster. Secondly the generation scheme presented in [Poldi 2009] generates bins with an average combined capacity of 9 – 14.5 times the sum of the item sizes on these instances, and as DPLS uses most bins in each subproblem, the running time will suffer considerably. The local search could perform better if the neighborhood was modified to take such instances into account, but the heuristic presented in [Haouari 2009a] has already shown promise on such instances.

[12] This scheme might generate infeasible instances. These have been proven infeasible and then discarded.

Class	0.25 sec.	0.5 sec.	1 sec.	5 sec.	10 sec.
1	**68.84**	68.54	66.96	61.58	60.81
2	**439.72**	430.22	421.09	413.50	412.18
3	**534.39**	526.29	515.96	505.99	500.37
4	**90.99**	87.78	80.70	71.69	65.68
5	1122.00	907.38	745.29	553.80	511.64
6	1075.65	849.27	732.72	644.49	609.42
7	**31.95**	29.43	27.52	19.95	15.02
8	**295.21**	250.06	218.28	187.79	177.96
9	**413.72**	314.77	270.29	243.19	233.29
10	**39.76**	37.77	35.70	29.45	25.98
11	804.37	502.18	341.48	210.20	183.75
12	963.58	606.29	429.47	308.78	276.30
13	**11.46**	10.50	7.73	3.05	1.73
14	329.54	**202.30**	149.24	96.06	88.78
15	433.12	322.32	**242.04**	150.84	141.48
16	**25.27**	23.97	22.895	15.13	14.36
17	949.96	608.61	386.89	**115.69**	93.00
18	1271.62	830.38	468.21	**138.14**	105.77

Table 2.4: Results of running DPLS on instances from Table 2.3 with varying limits on the running time. The bold font marks the earliest result that is better than that of *any* heuristic from [Poldi 2009]

2.8.3 Real Life Instances

The instances used in this chapter are artificially generated to simulate data similar in properties to the largest instances obtained from Danfoam.[13] The number m of different item sizes are varied from two to eight, two being the absolute minimum making sense, and every instance being trivial for the developed heuristics with more than eight item sizes, as all but the last bin typically are filled to optimality by the construction heuristic. The bin sizes are chosen with a uniform distribution between 2900 and 3050 centimetres for 70% of the bins, the rest are chosen from a uniform distribution between 800 and 2900. Enough bins are created to cut out twice the needed amount.

The heuristics performance on the generated instances shows the same characteristics that were observed on the more difficult instances of the real life problem. These instances can thus be used for a fair comparison between heuristics.

The algorithm is set to run for a maximum of ten seconds, but stopping if a zero waste solution is obtained. The average time usage in table 2.5 thus becomes an indicator for how often that is the case.

[13] This is done to make comparisons possible by providing generators on request, as the real instances are confidential.

n	m	DPC	DPLS	DPLSW	time DPC	time DPLS	time DPLSW
500	2	2186,10	153,40	145,45	0,0078	10,0015	10,0023
500	3	1340,11	9,76	9,43	0,0079	5,1965	5,0155
500	4	1563,57	6,99	1,75	0,0083	1,5303	1,4350
500	5	1538,22	0,28	0,19	0,0087	0,4522	0,4585
500	6	1343,01	0,00	0,00	0,0088	0,0490	0,0388
500	7	1352,97	0,00	0,00	0,0090	0,0523	0,0281
500	8	1392,54	0,00	0,00	0,0090	0,0232	0,0190
1000	2	3157,52	505,97	503,29	0,0288	10,0080	10,0153
1000	3	1910,57	66,03	58,52	0,0311	7,4589	7,7393
1000	4	1763,03	19,84	10,52	0,0406	2,8364	3,2317
1000	5	1554,60	0,74	0,82	0,0329	0,7969	1,0762
1000	6	1474,65	0,00	0,08	0,0344	0,2420	0,4270
1000	7	1446,43	0,09	0,07	0,0351	0,2620	0,2463
1000	8	1346,92	0,00	0,00	0,0359	0,1929	0,1431

Table 2.5: Average quality of solutions (in terms of units of waste) and time consumption (in seconds) for problems with n items and m different item sizes. Collected over 100 instances

It is clear from table 2.5 that the problem becomes easier as the number of item sizes increases. Further tests indicated that a large number of bin sizes have a positive effect as well. It is also clear that DPLSW usually produces solutions of a better quality than DPLS, though the limited gain on some instances might surprise. For the instances with a low number of items, the major part of the waste comes from the inability to fill any bin or most bins and the increased flexibility of ignoring waste on a bin is of limited use. For the instances with a high number of items, the waste is low enough for the randomness of the local searches to play a significant role.

Figure 2.1 and 2.2 shows that the local search heuristic usually converges relatively fast, but that it can be prone to random delays as is seen for the run on $n = 1000$ and $m = 8$. This is expected to be a result of optimizations of particularly expensive but irrelevant subproblems, and can be addressed using more finely tuned traversal of the neighbourhood, or by traversing it multithreaded. The initial solutions to the local search usually has more than 1000 and sometimes up to 4000 units of waste. To keep the scale in check, this is shown by a line extending beyond 1000 units of waste. If the algorithm ever encounters a solution with no waste, the line will disappear before the tenth second.

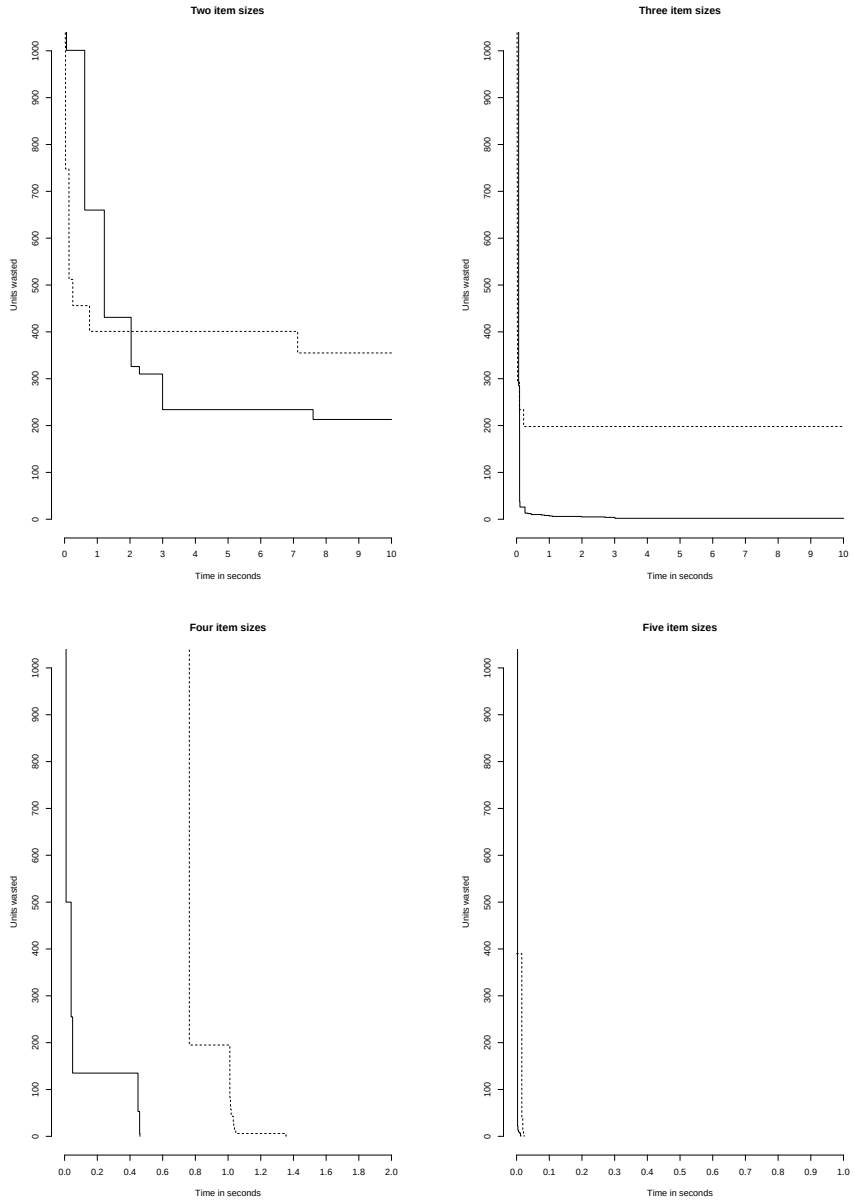

Figure 2.1: An example of solution quality as a function of time, solid lines and dashed lines being instances with $n = 500$ and $n = 1000$ respectively.

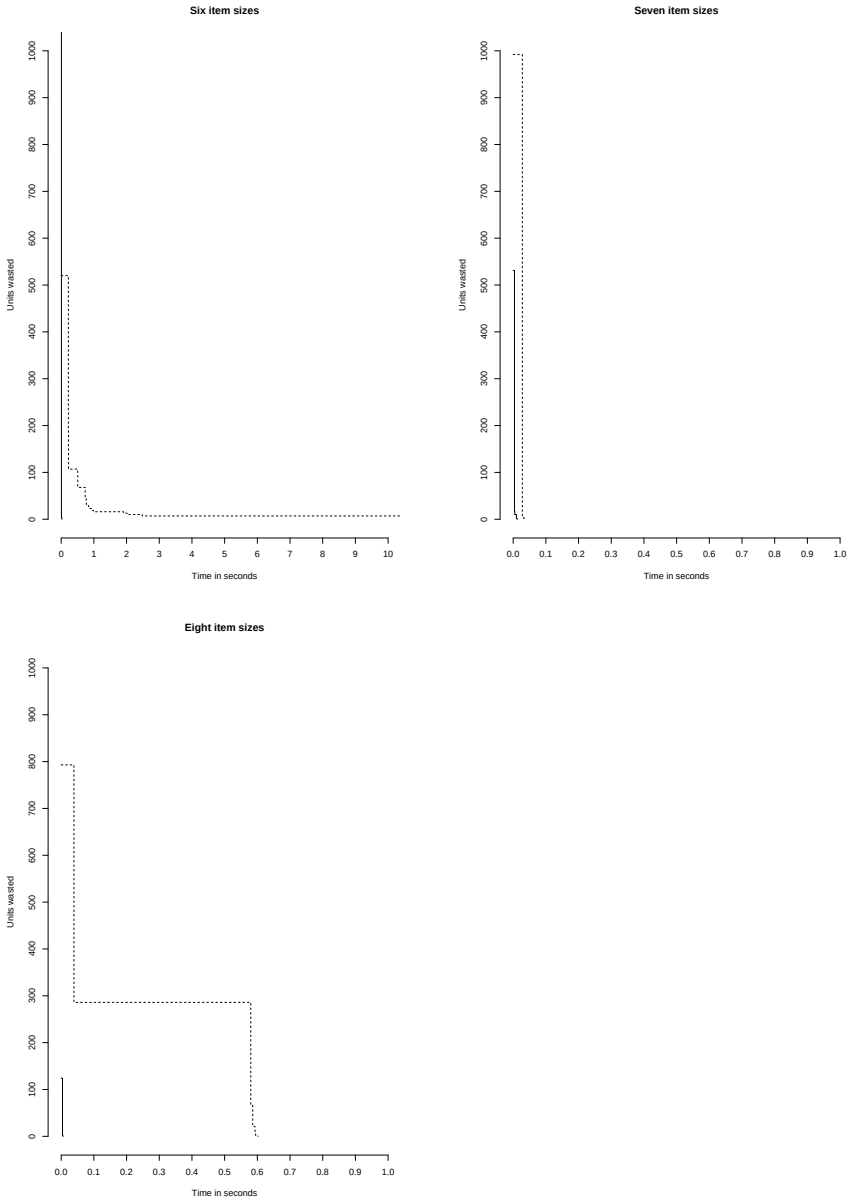

Figure 2.2: An example of solution quality as a function of time, solid lines and dashed lines being instances with $n = 500$ and $n = 1000$ respectively.

2.9 Adaptability of the Solver

One of the strengths of the local search heuristic is that it is often flexible in what problems and constraints it is able to handle. Therefore its handling of the following properties should be summed up:

Input minimization vs. output maximization. DPLS is made to handle input minimization problems, but as noted in [Wäscher 2007] fairly simple schemes exists for transforming output maximization problems into these.

Objective function. Currently DPLS handles only problems with a one to one correlation between the objective function and the amount of waste on bins, but it has the possibility to deduct a predescribed number of or all usable leftovers.

Homogeneity of items and objects. DPLS generally performs better the more heterogeneous objects and items are, but it has an acceptable performance on problems with homogeneous object and item sizes as well.

Average number of items per object. DPLS currently performs better on problems having a high number of items per object. Some of its deficiencies on instances with a low average number of items per object might be mitigated by further work, but other methods from the literature performs exceedingly well on those.

Time for solving. DPLS can produce reasonably acceptable solutions within half a second on all benchmark instances, and it can potentially use any amount of time available to improve on these until optimality is reached.

The proposed local search heuristic has proven flexible, and as its representation of data is very straightforward, additional constraints that might arise, should often been straightforward to implement.

2.10 Real Life Lessons

This subsection summarizes the real life constraints and concerns encountered when doing this work.

2.10.1 Optimized, Actual and Perceived Quality

It is well known that objective functions can be approximations to the true cost incurred by the company employing the optimization method. In Danfoams case exogenous constraints arose from a subsequent processing inferring a "slicing height" on each item, and punishing each change of slicing height when evaluating a solution. Furthermore changes to item sizes were considered unfavourable.

The additional objectives turned out to be algorithmically uninteresting, as a strict preference a priori was given to minimizing wasted material, then to minimizing changes in slicing height and lastly minimizing changes in item sizes. This made the problem solvable by post processing: First an initial sort based on slicing height, then rearranging slicing heights to match between blocks if possible. Lastly items having the same slicing height were sorted by size, and blocks of items with equal slicing height could be rearranged to eliminate changes in item sizes. This process did not guarantee optimality, but produced good results in practice.

Even when the proposed solution is actually optimal it might not be perceived as such by the operators of the machinery. In the Danfoam case this would be the case when few item sizes were present and a very good solution was obtainable by exhausting a certain item size making the residual problem much harder. If such scenarios are seen by workers having only capacity to consider the next block/bin, a deviation from the proposed solution may occur. This scenario can be avoided by multiple approaches:

- Strictly enforcing the execution of a plan with the risk of alienating the workers. The feasibility of this approach depends on the work culture of the company.

- Educating the worker on possible impacts on residual problems, which becomes infeasible for all but the simpler categories of problems.

- Creating a visual interface whereby the workers can see the solution as a whole, realising that any modification to the proposed solution has ramifications with respect to the rest of the solution.

In the Danfoam case such as visualization tool was developed, and also helped to visualize the improvements obtained by the developed heuristics.

2.10.2 "Good Enough"

Customers will usually pay a fee proportional to the time spent developing real life applications of optimization methods. This naturally causes an upper bound on the complexity of the optimization method, where development costs would make the return on investment too unattractive if not the whole project unprofitable. A second bound is encountered when the optimization of a problem reaches a state where other problems in the company dominates the problem under consideration in terms of return on investment.

Further studies into stochastic modelling and robust optimization were proposed, but Danfoam lost economic interest in the project when the performance of DPLSW was presented, as the waste dropped below what they considered an acceptable margin.

2.10.3 Time Constraints

Time constraints are well known and in the literature, where they are typically represented as hard constraints. This corresponds well with the problem presented by the anonymous company, as the optimization method is used internally in a fast paced machinery. Danfoam however uses the optimization methods via a human interface, and while they have a fuzzy limit on the waiting time they allow, this limit can be bent if the achieved benefit is deemed significant enough. As these limits are not quantifiable by the Danfoam, and might even be situational, reconfigurability becomes important. Not only by providing a solution at "anytime", but converging fast enough for that solution to be usable.

2.10.4 Stochasticity

The paramount concern encountered was the presence of stochasticity. Uncertain information is often present, represented as a probability distribution describing the uncertain event. Danfoam considered the possibility of obtaining better estimates of the lengths of blocks in the storage area, by investing in measuring equipment and having employees perform the measurements. The idea was rejected due to the waste realised after using DPLSW was too low to justify the upkeep. This does however indicate that stochasticity can be minimized by investments under certain circumstances.

2.11 Conclusions

A construction heuristic for the variable size bin packing problem has been described, and shown to perform well, also given the larger bin sizes and item to bin ratio than in previous applications of similar methods. Its use is recommended in all but the very most time or memory critical applications, or instances using *very* large bin sizes.

Methods based on the construction heuristic efficiently solved a real life problem involving very fast machinery, and deadlines measured in milliseconds. This allowed the anonymous company to increase the number of scales in the machine from the 16 to 32 that was their previous maximum, up to 64 scales which corresponds to their maximum produced capacity. Previously the 64 scales machines were compartmentalized into 16 or 32 scale sections leading to suboptimal packaging.

Furthermore a local search heuristic was presented, that scaled well with the number of different bin and item sizes as well as the number of items per bin, at the cost of a poorer scaling with the maximal bin size. The methods also allowed optimization to be resumed at any time, given changes in input, without any ramp up time. Additionally the method allowed for the waste on the worst packed bin to be ignored in a straightforward manner. These properties supplements the methods presented in [Cui 2010], [Cherri 2009], [Haouari 2009a] and [Poldi 2009]. The DPLS heuristic furthermore outperformed the solvers presented in [Poldi 2009] on a large

subset of the instances.

The local search heuristic was applied to a real life problem, and improved material usage in Danfoam significantly.

In chapter 4 the lessons learned during the real life application will be used during the development of the dynamic rescheduling framework.

Chapter 3

Scheduling Outages for Nuclear Power Plants

Contents

3.1 Contribution of the Author

This chapter is an extended version of the paper [Godskesen 2011], done in collaboration with the Ph.D. students Steffen Godskesen, Thomas Sejr Jensen and Niels Kjeldsen. Most of the work used in the paper is a product of several iterations of ideas and adaptations, and all contributors are thus intimately familiar with the concepts described herein, and have contributed to their development.

The main contributions of the author is the original NP-completeness proof, the reinsertion heuristic and some of the data structures key to the performance of the heuristics developed.

3.2 Introduction

In France, the majority of the electricity is produced by thermal — and in particular nuclear — power plants, and the main supplier is Électricité de France. There are two types of thermal power plants in the portfolio: *type 1 plants* which can be supplied with fuel continuously and without interrupting the production, and *type 2 plants* (nuclear power plants) which must be taken offline for refuelling at regular intervals. The process of taking a power plant offline for maintenance is called an *outage* and the period between two outages a *production campaign*. While type 1 plants are more flexible than type 2 plants, the production cost incurred per unit of electricity is larger than for type 2 plants. Outages for type 2 plants should be scheduled such that the demand for electricity is satisfied at the lowest possible cost. A schedule for outages must satisfy a large number of constraints due to for example limited resources and safety considerations. Some of the constraints are due to limits on fuel levels in connection with each outage: There is a maximal fuel level for a power plant, a maximal allowed fuel level when a plant is taken offline, and a minimum refuelling amount at each outage. An outage lasts a couple of weeks and is always scheduled to begin at the start of a week.

When planning future production, one must decide i) the timing of each outage, ii) refuel amounts, and iii) electricity production levels. The total production of electricity is not allowed to exceed the demand, and therefore type 2 plants can sometimes produce at less than their maximum production level. This situation is called *modulation*, and due to technical reasons there is a limit on the allowed modulation for each plant for each production campaign. By modulating nuclear power plants, one can adjust to small variations in electricity demand. Time is discretised into *time steps* spanning a couple of hours to model the variations in demand.

When a type 2 plant is taken offline for refuelling and maintenance, the electricity must be produced by alternative sources. This can either be done by one of the other type 2 plants or by the more expensive type 1 power plants. The future electricity demand and the price of fuel for production on type 1 plants is uncertain. This uncertainty should be taken into account in the planning of outages for type 2 plants. This is modelled by minimizing the *expectation* of future production cost over a number of given scenarios. The decisions on outage placement and refuelling are common across all scenarios, whereas the decisions on production can be adjusted for individual scenarios.

3.2.1 The ROADEF/EURO Challenge 2010

The described problem was the subject of the ROADEF/EURO Challenge 2010, which ran from July 2009 through June 2010. The competition was jointly organized by the French Operational Research and Decision Support Society, the European Operational Research Society, and the European utility company Électricité de France. In total 44 teams from 25 countries signed up for the challenge, of these 21 qualified for the final round, and 16 submitted a program for the final evaluation. The submitted programs were evaluated on ten problem instances, five known in advance and five only used in the evaluation. For each instance the time limit imposed on the program was one hour. In the concluding phase of the competition our team ranked second in the junior category and sixth overall. A complete description of the competition and evaluation rules can be found in [Porcheron 2010].

The problem instances from the challenge have up to 75 type 2 plants, up to 120 scenarios for future prices and demand, and up to 5817 discrete time steps. Outages always start at the beginning of a week, and since the time horizon is about five years, there can be up to 277 possible outage start dates. This leads to a large-scale energy management problem.

A solution to the problem can be divided into two parts: a *maintenance schedule* and a *production plan*. A maintenance schedule specifies when outages of type 2 plants are scheduled and the amount of fuel to reload at each outage. A production plan specifies the production level of each plant for every time step and every scenario. To give an example of what a maintenance schedule might look like, see Figure 3.1, which shows a schedule for 11 type 2 plants.

Figures 3.2 and 3.3 give an example of production level and fuel level over time for plant number 8. Note the sudden decrease in production around time step 2200; this is modulation to ensure that there is no overproduction in the specific time step.

The following factors indicate that the problem is computationally hard to solve. First of all, it is NP-hard in the strong sense, as we prove in Section 3.3.5. Furthermore, the problem instances are large: a single problem instance takes up to 262 megabytes of hard disk space and contains more than $50 \cdot 10^6$ continuous decision variables just to represent production levels for every plant, time step, and scenario. Finally, there is a large number of constraints on production levels, fuel

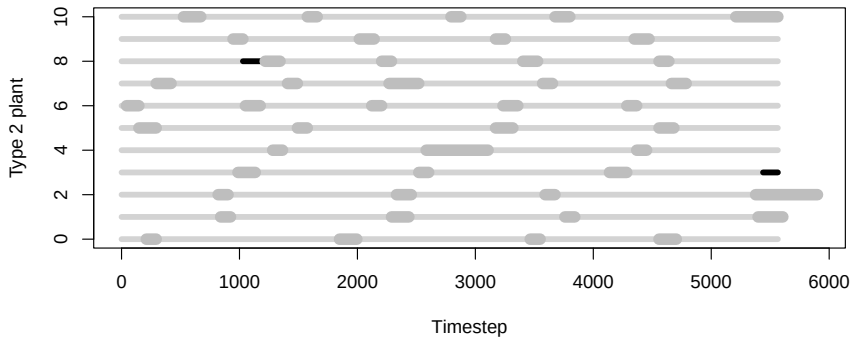

Figure 3.1: A maintenance schedule for 11 type 2 plants. A light gray line is used for time steps with production, a wide dark gray segment for outages, and a black segment for when a plant is out of fuel.

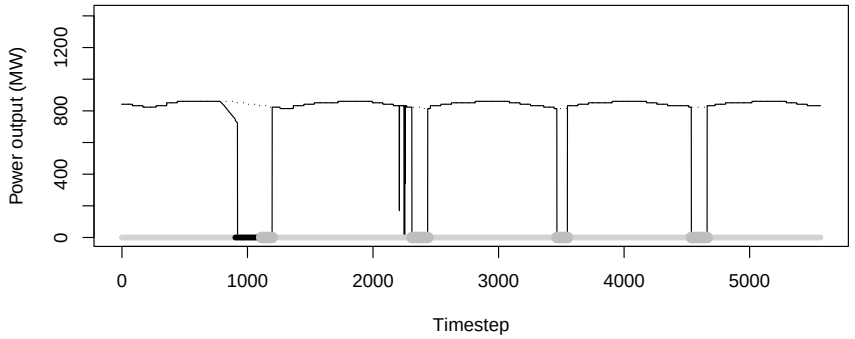

Figure 3.2: The production level over time for plant 8 from Figure 3.1. The full line is the current production level and the dashed line the maximum allowed production level.

levels, refuelling amounts, and scheduling of outages.

The program we submitted contained a small implementation bug which resulted in infeasible solutions for two of the five unknown instances (feasible solutions were found for all other instances). After correcting the program, we are able to find feasible solutions for all instances, and the solutions are actually better than those of the winning team.

3.2.2 Related Work

A problem similar to the one studied here has been considered in [Fourcade 1997]. They consider roughly the same scheduling problem as here and formulate a mixed integer programming model. In their model, there is no decision variable concerning refuelling amounts; this decision is instead handled as a predefined fixed amount. There is also no uncertainty of future demand and prices, and the demand is given per week, in contrast to the competition where the electricity demand is given per time step, i.e., their discretisation is more coarse-grained.

They are able to solve small problems of up to 20 nuclear power plants with a

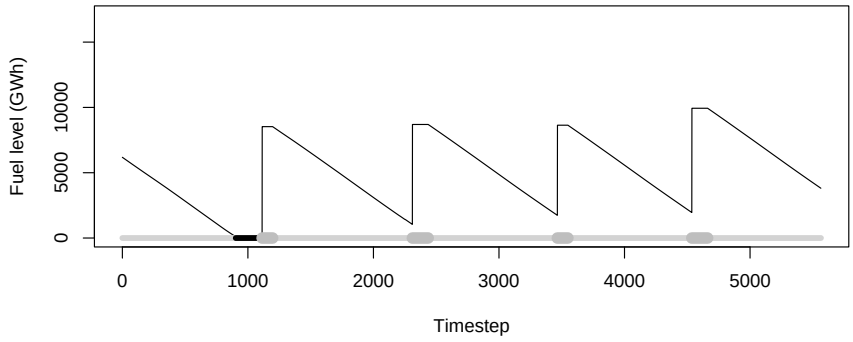

Figure 3.3: The fuel level over time for plant 8 from Figure 3.1.

MIP solver. The authors also attempted to tackle a model with 40 power plants, which yields a feasible solution after more than eight hours of computational time but with a significant gap to the LP lower bound. They report that this model is almost the size of the complete model of France which has 54 nuclear plants. The performance of MIP solvers and hardware have increased significantly since 1997, so modelling that particular problem as a mixed integer programming problem might be feasible with state of the art solvers.

Besides the work by Fourcade et al. very little is published on the topic. Nuclear maintenance and refuelling is mentioned by [Dunning 2001], but the objective considered is to minimize the environmental impact.

The scheduling part of this problem is similar to the Resource Constrained Project Scheduling Problem (RCPSP), see e.g. [Neumann 2003]. In both problems, activities (in this case outages) have to be scheduled subject to temporal constraints and limited resources. However, the problem at hand includes several constraints not found in common variants of RCPSP, such as disjunctive temporal constraints that only apply if a pair of activities is scheduled in a specified interval (as described in further detail in Section 3.3.3).

Setting production levels for power plants is treated in the literature under the term 'economic dispatch' — i.e., the problem of dispatching units to producing power in an economic way. While many settings have been considered, see for example [Chowdhury 2002], there are new features in the production planning for nuclear power plants. These features concern special bounds on production levels when the fuel level is low, which lead to nonlinear constraints. When the fuel level of a type 2 plant drops below a given threshold, a predetermined decreasing production profile is imposed. Without this constraint the production planning could be solved as a linear programming problem.

3.2.3 Our Contributions

In this chapter we prove that the considered energy management problem is NP complete in the strong sense, and propose a three phase hybrid optimization approach to obtain high quality solutions to the problem. Hybrid methods are pop-

ular in recent literature, see [Blum 2008, van Hentenryck 2011, Hooker 2011]. The proposed method is a three-phase heuristic approach for the management problem introduced above. The key parts of our approach are an initial solution construction and a two-part solution improvement phase. A similar decomposition was also present in the best performing approach of the previous ROADEF competition, see [Bisaillon 2009].

The proposed hybrid approach consists of a constraint programming (CP) model in which an initial solution to the complex scheduling problem is found by using approximated constraints for production levels and fuel consumption. From this first schedule we apply a stochastic local search (SLS) algorithm based on a simple neighbourhood structure. Two essential components in the local search are a very fast feasibility check and a fast but approximated evaluation of the change in solution cost. To guide the search, we use a simulated annealing metaheuristic. In the third and final phase we use a greedy algorithm to remove any overproduction.

The approach is the result of an engineering cycle in which we recognized that the SLS had difficulties finding an initial feasible solution, while the task turned out to be easy when tackled by CP. On the other side, the CP was not able to improve the solution with respect to the objective function, something the SLS turned out to be more successful in doing.

Other elements that turned out to be important are the aggregation and approximation of the production planning in both the CP model and the local search.

3.2.4 Overview

The chapter is organized as follows. Section 3.3 gives a formal description of the optimization problem as well as a complexity proof. In Section 3.4 we describe how to obtain a feasible solution using a CP model for scheduling outages and a heuristic for production planning. The stochastic local search phase is described in Section 3.5. Section 3.7 describes how overproduction is handled. Computational analysis and results are the topic of Section 3.8. Finally, we conclude and give directions for further research in Section 3.9.

3.3 Problem Description

Each type 2 plant goes through a number of *cycles*. A cycle is composed of an outage followed by a production campaign. During an outage the plant cannot produce electricity because of maintenance and reloading of fuel. During a production campaign the plant can produce electricity. Having type 2 plants producing at less than maximum capacity leads to wear of the equipment involved and should thus be avoided if possible. The difference between maximum capacity and actual production is called modulation.

The demand for electricity is not known with certainty at the time of planning. This uncertainty is modelled by introducing a number of *scenarios*, each of which represents a realistic future demand profile for the planning horizon discretised into

a number of time steps. Optimizing for several realistic scenarios instead of just one generally leads to more robust plans.

Decisions concerning scheduling of outages and refuelling amounts must be shared by all scenarios, but production levels can be determined for each individual scenario. This creates a dependency between scenarios, since the outage schedule and refuelling amounts must be feasible with respect to every scenario's production plan.

The objective is to create both a maintenance plan and production plan that satisfy the demand for electricity at the lowest average cost over all scenarios. The cost must be minimized while satisfying a number of constraints. These constraints can be divided into four categories: i) bounds on production levels, ii) bounds on fuel levels and refuelling amounts, iii) different kinds of temporal constraints on the scheduling of outages, and iv) bounds on the outages' simultaneous use of limited resources.

3.3.1 Decision Variables and Bounds

We use $s = 0, \ldots, S-1$ to index scenarios, $t = 0, \ldots, T-1$ to index time steps, $h = 0, \ldots, H-1$ to index weeks, $j = 0, \ldots, J-1$ to index type 1 plants, $i = 0, \ldots, I-1$ to index type 2 plants, and $k = 0, \ldots, K-1$ to index cycles. A week consists of a number of time steps, i.e., two different discretisations of the planning horizon are used. This is because outages are scheduled on a weekly basis, whereas a higher resolution is required for the productions levels. The length of a time step in hours is denoted by D (all time steps have the same length).

The length of outage k for type 2 plant i is denoted by $DA_{i,k}$. Let $ha(i,k) \in \mathbb{Z}$ denote the week that the k'th outage for type 2 plant i starts, and $TO_{i,k}$ and $TA_{i,k}$ denote the earliest and latest starting time, respectively, for $ha(i,k)$, formally

$$TO_{i,k} \leq ha(i,k) \leq TA_{i,k}. \tag{3.1}$$

The bounds $TO_{i,k}$ and $TA_{i,k}$ may be undefined, in which case the corresponding inequality is trivially satisfied. If the upper bound is undefined for some outage, the outage does not have to be scheduled. If outage k for plant i is not scheduled then $ha(i,k) = -1$ and constraint (3.1) is not enforced. Outage $k+1$ for plant i cannot start before outage k for plant i is finished, and outage $k+1$ cannot be scheduled unless k is scheduled.

The amount of fuel reloaded at type 2 plant i in outage k is denoted by $r(i,k) \geq 0$ and if k is scheduled $r(i,k)$ must satisfy

$$RMIN_{i,k} \leq r(i,k) \leq RMAX_{i,k}, \tag{3.2}$$

where the bounds $RMIN_{i,k}$ and $RMAX_{i,k}$ are input data. If k is not scheduled, then $r(i,k) = 0$.

Let $p(\ell, t, s) \geq 0$ denote the production of plant ℓ (which may be of type 1 or 2) at time step t in scenario s.[1]

[1] As in the original problem formulation [Porcheron 2010] we index decision variables using

3.3.2 Auxiliary Variables

In addition to the decision variables there is a number of auxiliary variables which can be derived from the decision variables and thus do not increase the size of the solution space.

The set of time steps composing the k'th outage of type 2 plant i is denoted by $ea(i,k)$, and the set of time steps composing the subsequent production campaign is denoted by $ec(i,k)$. For any outage k, the production $p(i,t,s)$ of plant i must be zero for every $t \in ea(i,k)$ and every scenario s.

The fuel stock of type 2 plant i at time step t in scenario s is denoted by $x(i,t,s) \geq 0$. The initial fuel level of plant i (at time step 0) is denoted by XI_i and specified in the input data. During a production campaign for plant i the decrease in fuel level from time step t to $t+1$ in scenario s equals the production multiplied by the length of a time step D, and therefore

$$x(i,t+1,s) = x(i,t,s) - p(i,t,s)D. \tag{3.3}$$

During an outage the fuel level at type 2 plant i increases because of refuelling. Due to technical reasons the new fuel level is not simply the sum of the old fuel level and the amount reloaded. Formally, if t is the first time step in outage k for plant i, then the new fuel level for i in scenario s is computed by

$$x(i,t+1,s) = Q_{i,k}x(i,t,s) + r(i,k) + Q'_{i,k}, \tag{3.4}$$

where $Q_{i,k} < 1$ and $Q'_{i,k}$ are input data[2].

3.3.3 Constraints

The problem constraints are here divided into four groups, namely production level constraints, fuel level constraints, scheduling constraints, and resource constraints.

Production level constraints Let $DEM^{t,s}$ denote the *demand* at time step t in scenario s. The total production must equal the demand in every scenario and every time step

$$\forall s,t : \sum_{j=0}^{J-1} p(j,t,s) + \sum_{i=0}^{I-1} p(i,t,s) = DEM^{t,s}. \tag{3.5}$$

Let $PMIN_j^{t,s}$ and $PMAX_j^{t,s}$ denote the minimum and maximum, respectively, *allowed production* of type 1 plant j at time step t in scenario s, then

$$\forall s,t,j : PMIN_j^{t,s} \leq p(j,t,s) \leq PMAX_j^{t,s}. \tag{3.6}$$

parentheses in order to distinguish them from parameters in the model.

[2]Equation (3.4) is a simplification of Equation (CT10) in the original model defined by ROADEF, but the two formulas are equivalent when appropriately adjusted values for $Q_{i,k}$ and $Q'_{i,k}$ are used.

The bounds on production for a type 2 plant are more complex, since they depend on the current fuel stock of the plant. If the fuel level is above a specified threshold $BO_{i,k}$, then the production is bounded from above by $PMAX_i^t$

$$\begin{aligned} \forall s,t,i,k : t \in ec(i,k) \wedge x(i,t,s) \geq BO_{i,k} \\ \Rightarrow 0 \leq p(i,t,s) \leq PMAX_i^t. \end{aligned} \tag{3.7}$$

As long as the fuel level is above the threshold modulation is undesirable and therefore there is an upper bound $MMAX_{i,k}$ on the accumulated modulation of plant i in each production campaign k

$$\forall s,i,k : \sum_{\substack{t \in ec(i,k) \wedge \\ x(i,t,s) \geq BO_{i,k}}} (PMAX_i^t - p(i,t,s))D \leq MMAX_{i,k}. \tag{3.8}$$

If the fuel level is below the threshold $BO_{i,k}$, the upper bound decreases and a lower bound is also enforced. This is referred to as the *declining power profile*. How much the upper bound decreases for type 2 plant i in production campaign k is specified by the function $PB_{i,k}$ which maps fuel levels to real numbers between zero and one. Formally, for all s,t,i,k, if $t \in ec(i,k)$ and $x(i,t,s) < BO_{i,k}$, then the production must lie in a small interval centered around P_x

$$P_x = PB_{i,k}(x(i,t,s)) \cdot PMAX_i^t, \tag{3.9}$$

$$(1-\varepsilon)P_x \leq p(i,t,s) \leq (1+\varepsilon)P_x. \tag{3.10}$$

Note that if the plant produces at P_x and this implies that the plant runs out of fuel, it is not allowed to produce at all. Thus, (3.10) applies only if

$$x(i,t,s) \geq P_x \cdot D. \tag{3.11}$$

If (3.11) does not hold, $p(i,t,s)$ must be zero.

Fuel level constraints There are upper bounds on the *fuel level* before and after a type 2 plant's outage. Let $AMAX_{i,k}$ denote the upper bound on the fuel level at the time when outage k for plant i starts and $SMAX_{i,k}$ the upper bound on the fuel level after the outage k for plant i. If the k'th outage for plant i starts at time step t, inequality (3.12) and (3.13) must hold for every scenario s

$$x(i,t,s) \leq AMAX_{i,k}, \tag{3.12}$$

$$x(i,t+1,s) \leq SMAX_{i,k}. \tag{3.13}$$

Scheduling constraints There are disjunctive temporal constraints between outages. If a specified pair of outages (i,k) and (i',k') is scheduled such that both intersect a given interval (this interval may be the entire planning horizon), then the following constraint must be satisfied:

$$ha(i,k) - ha(i',k') \geq Se \vee ha(i',k') - ha(i,k) \geq Se', \tag{3.14}$$

where the lower bounds Se and Se' are input data.

Several types of temporal constraints are defined in the original problem definition from ROADEF [Porcheron 2010], but they can all be converted to the type in (3.14).

Resource constraints For every week h there is a collection of subsets of outages. For each such subset A and an associated natural number N, at most N of the outages in A are allowed to be on outage in week h:

$$\forall h : \sum_{(i,k)\in A} \Phi(i,k,h) \leq N, \tag{3.15}$$

where $\Phi(i,k,h)$ equals 1 if plant i is in outage k in week h and 0 otherwise.

There are limited resources available for maintenance. To model this, a collection of subsets of outages is given. Each subset B in the collection has an associated resource availability Q. For every B, at most Q of the outages in B can use maintenance resources in any week

$$\forall h : \sum_{(i,k)\in B} \Phi(i,k,h) \leq Q, \tag{3.16}$$

where $\Phi(i,k,h)$ equals 1 if outage (i,k) uses a maintenance resource in week h and 0 otherwise. Note that the weeks in which an outage uses resources are not necessarily the same as the weeks in which it is in outage. This difference is specified in the input data.

Finally, there is an upper bound on the capacity that is allowed to be offline at a given time in a given region. To model this, a collection of subsets of plants is given. Each subset C in this collection has an associated upper bound $IMAX$ and a subset of weeks IT. For every C, $IMAX$, and any week h in IT, the total offline capacity of plants in C is not allowed to exceed $IMAX$

$$\forall h \in IT : \forall t \in h : \sum_{\substack{i\in C:\exists k:\\ t\in ea(i,k)}} PMAX_i^t \leq IMAX. \tag{3.17}$$

Note that in (3.17) a week is considered as a set of time steps. The sum is simply over all type 2 plants in C that are offline at time step t.

3.3.4 Objective Function

The objective function is composed of three terms: the total cost of refuelling all type 2 plants, the average cost of type 1 production over all scenarios, and the value of residual fuel at type 2 plants at the end of the planning horizon. Let $C_{i,k}$ denote the cost of fuel for type 2 plant i in cycle k, $C_{j,t,s}$ the cost of production for type 1 plant j at time step t in scenario s, and C_i the value of fuel for type 2 plant i at the

end of the planning horizon. Then the objective function to be minimized is

$$\sum_{i=0}^{I-1}\sum_{k=0}^{K-1} C_{i,k}r(i,k) + \frac{1}{S}\sum_{s=0}^{S-1}\sum_{t=0}^{T-1}\sum_{j=0}^{J-1} C_{j,t,s}p(j,t,s)D- \frac{1}{S}\sum_{s=0}^{S-1}\sum_{i=0}^{I-1} C_i x(i,T,s). \tag{3.18}$$

3.3.5 Computational Complexity

To prove the NP-hardness of the problem under consideration, we propose a reduction from 1-in-3-SAT, which is proved to be NP-hard in the strong sense by [Schaefer 1978]. Reductions directly from a scheduling problem might be possible but is complicated by the exponential (albeit pseudo-polynomial) number of time steps that often will arise.

Given a boolean formula $\beta_1 \wedge \cdots \wedge \beta_c$ where each clause β_i, $1 \leq i \leq c$, is the disjunction of three boolean literals from the set $\{x_1, \ldots, x_n\}$, 1-in-3-SAT asks for an assignment of true or false to $x_1, \ldots, x_n$ such that exactly one of the literals in each clause β_i evaluates to true.

To solve an instance of 1-in-3-SAT by using the problem under consideration, we construct an instance having a single scenario as follows. A type 2 plant i with a single outage with a duration of one week is created for each clause β_i. Furthermore, a week is created for each variable x_h and its negation $\neg x_h$, in such a way that the variable and its negation occupy successive weeks. Scheduling an outage in the week that corresponds to x_h (respectively $\neg x_h$) is interpreted as forcing x_h to be true (respectively false).

All outages must be scheduled in order for the 1-in-3-SAT instance to be satisfiable. A constraint of type (3.1) with bounds set to include all $2n$ weeks for every outage ensures this.

To ensure that the single outage for plant i can only be scheduled in one of the three weeks corresponding to literals in β_i, a constraint of type (3.17) which restricts the amount of offline capacity is added for the single outage. We set $PMAX_i^t = 1$ for all t and $IMAX = 0$. Furthermore, we let C contain plant i's single outage and let IT contain all weeks except the three corresponding to literals in the β_i. Constraint (3.17) is enforced on time steps rather than weeks, so the number of time steps is set to $2n$ such that there is one time step per week.

Constraints of type (3.14) are added to ensure that outages are not scheduled in both the week corresponding to x_h and $\neg x_h$. For each pair of clauses that contains literal x_h and $\neg x_h$ respectively, a constraint of type (3.14) is defined on the two weeks corresponding to x_h and $\neg x_h$ (which are consecutive by construction). Forcing a separation of two weeks between the corresponding plants' outages prevents these outages from being scheduled in the weeks corresponding to x_h and $\neg x_h$, respectively.

See Figure 3.4 for an example of a simple boolean formula encoded as a maintenance scheduling problem.

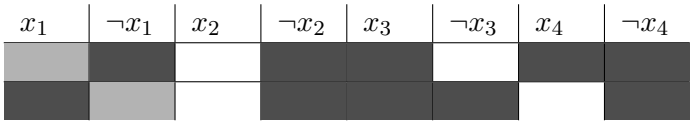

Figure 3.4: A representation of the formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$. There is one row per clause in the formula. Dark gray weeks are disallowed using constraint (3.17). Outages can be scheduled in light gray and white weeks, but the light gray weeks for x_1 and $\neg x_1$ cannot both be used for outages due to a constraint of type (3.14).

The construction described above is polynomial in the input size, as we have c clauses, giving rise to c outages, each with three valid weeks where it can be scheduled. For each of these weeks, we need less than c constraints to restrict it from conflicting with the other outages. The conversion is thus bounded from above by $3c^2$.

The remaining constraints are constructed such that they do not prevent scheduling of any type 2 plant in any week. A single type 1 plant can be used to cover any demand we decide on. We set the demand $DEM^{t,0} = I$ for each time step t, and choose $PMAX_i^t = 1$, $PMIN_i^t = 0$ for each type 2 plant i and time step t. The initial fuel stock is set large enough to allow every type 2 plant to produce in all $2n$ weeks without outages. Minimum and maximum bounds on refuelling of the type 2 plants are set such that they do not constrain the scheduling solution, i.e., $RMIN_{i,k} = 0$, $RMAX_{i,k} = 0$. To ensure that $AMAX_{i,k}$ and $SMAX_{i,k}$ do not become constraining, they are set to the initial fuel stock.

If and only if we can schedule all outages in the energy management problem, the 1-in-3 SAT instance is satisfiable. Truth values are assigned to literals in the given 1-in-3-SAT instance as follows. A literal is set to true if some outage is scheduled in the corresponding week and to false otherwise. Thus, any instance of 1-in-3-SAT can be solved by scheduling outages. As mentioned above, the size of the reduction's output is polynomial in the size of the 1-in-3-SAT instance and can obviously be constructed in polynomial time, and therefore the optimization problem in this chapter is NP-hard in the strong sense, i.e., it is NP-hard even if all numerical parameters are encoded in unary base.

3.3.6 Hybrid Approach Overview

Our solver constructs a solution in three phases.

1. In the first phase, solving a CP model creates an initial maintenance schedule with decisions on starting week and refuelling amount for each of the scheduled outages. The CP model does not include any decisions on production levels for type 1 or type 2 plants.

2. In the SLS phase of the approach, the maintenance schedule is improved (both the placement of outages and the refuelling amounts). The SLS uses the initial

maintenance schedule but recalculates the refuelling amounts. During the local search, production levels for the type 2 plants are the same for all scenarios. This is done to speed up evaluation of the quality of the current maintenance schedule. The evaluation is an approximation since calculating the true cost for all scenarios is computationally too expensive to be used in the local search procedure.

3. In the final phase, production levels for both type 1 and 2 plants for every scenario are calculated. In this phase the production levels of type 2 plants are also adjusted to remove overproduction if any is present.

Note that a full solution with production levels for every scenario only is available at the very end of the solution process.

3.4 Initial Solution Construction

We showed that the maintenance scheduling part of the problem is NP-hard, and our experience is that finding non-trivial feasible solutions to this problem is challenging in practice as well. Our initial approach included different directions: a set of construction heuristics combined with SLS and a CP approach. Only the CP approach consistently gave feasible maintenance schedules. Our overall setup thus starts by creating a first feasible maintenance schedule using CP.

3.4.1 Constraint Programming

An exact representation of the problem would result in a very large number of variables, since it requires modelling of every time step. To reduce the size of the model, we aggregate all data indexed by time steps into weekly equivalents. Furthermore, concepts such as modulation, the decreasing power profile, and the cost of type 1 production lead to an excessively large model. Therefore, we focus primarily on finding a feasible maintenance schedule and introduce a surrogate objective function that approximates the actual objective function (3.18), thereby leaving the rest of the optimization to the subsequent SLS.

The surrogate objective function guides the solver towards a solution with an appropriate number of outages. Too few outages will make the type 2 plants run out of fuel and unproductive for an extended period of time. Too many outages take the plant offline unnecessarily and might cause infeasibility, as the plants are unable to use enough fuel for the next outage to take place.

A CP model is used to find a feasible maintenance schedule.[3] For every outage i, k, the CP model has three decision variables: A binary variable $\sigma(i, k)$ deciding if outage k for type 2 plant i is scheduled or not, the integer variable $ha(i, k)$ determining the starting week for the outage, and the integer variable $r(i, k)$ determining the refuelling level. Refuelling amounts are continuous in the problem formulation

[3] For a general introduction to CP, see [Apt 2003].

but are discretised because most CP solvers cannot handle continuous variables. To reduce the domain of the refuel variables in the CP model, the discretisation is into segments of 1000 fuel units.

We model the scheduling and resource constraints (3.14)-(3.17) exactly but approximate the fuel level constraints because an exact representation requires exact modeling of fuel consumption of type 2 plants, which would lead to a very large model, due to the many time steps. The complete CP model is specified in the four subsections below.

3.4.1.1 Scheduling Constraints

The following two sets of constraints enforce the time windows defined in (3.1) on outages. For every outage (i,k) with lower bound $TO_{i,k}$ defined

$$\sigma(i,k) = 1 \Rightarrow ha(i,k) \geq TO_{i,k}. \tag{3.19}$$

For every outage (i,k) with upper bound $TA_{i,k}$ defined

$$\sigma(i,k) = 1 \Rightarrow ha(i,k) \leq TA_{i,k}. \tag{3.20}$$

For every outage (i,k) with upper bound $TA_{i,k}$ defined, we enforce the following constraint since (i,k) must be scheduled

$$\sigma(i,k) = 1. \tag{3.21}$$

To ensure that outages are scheduled in order, we enforce the following constraint for each type 2 plant i and every outage $k = 1,\ldots,K-1$

$$ha(i,k-1) + DA_{i,k-1} \leq ha(i,k), \tag{3.22}$$

where $DA_{i,k}$ is the length of outage i,k. Since scheduling of an outage requires scheduling of all previous outages for the same type 2 plant, we enforce the following constraint for each type 2 plant i and every outage $k = 1,\ldots,K-1$

$$\sigma(i,k) \leq \sigma(i,k-1). \tag{3.23}$$

To enforce the temporal constraints in (3.14), we add the following constraint for every given pair of outages (i,k) and (i',k'), interval of weeks $[L,U]$, and lower separation bounds Se and Se' in weeks

$$\begin{aligned}
&\sigma(i,k) = 1 \wedge \sigma(i',k') = 1 \wedge \\
&L < ha(i,k) + DA_{i,k} \wedge ha(i,k) \leq U \wedge \\
&L < ha(i',k') + DA_{i',k'} \wedge ha(i',k') \leq U \Rightarrow \\
&ha(i,k) - ha(i',k') \geq Se \vee ha(i',k') - ha(i,k) \geq Se'.
\end{aligned} \tag{3.24}$$

3.4.1.2 Resource Constraints

To ensure that at most a specified number of type 2 plants are offline in a week h, we implement constraints as in (3.15) with $\Phi(i,k,h) = 1$ if

$$\sigma(i,k) = 1 \wedge ha(i,k) \leq h \wedge h \leq ha(i,k) + DA_{i,k}, \tag{3.25}$$

and $\Phi(i,k,h) = 0$ otherwise.

To ensure that enough resources are available in every week, we implement constraints as in (3.16) with $\Phi(i,k,h) = 1$ if

$$\sigma(i,k) = 1 \wedge ha(i,k) + LU_{i,k} \leq h < ha(i,k) + LU_{i,k} + TU_{i,k}, \tag{3.26}$$

and $\Phi(i,k,h) = 0$ otherwise. Here, since the weeks in which an outage (i,k) uses resources are not necessarily identical to the weeks in which it is offline, $LU_{i,k}$ and $TU_{i,k}$ specify the resource usage's offset and duration, respectively.

To ensure that there is sufficient type 2 capacity online, we implement constraints based on (3.17) as follows

$$\forall h \in IT : \sum_{i \in C} \sum_{k=0}^{K-1} \Phi(i,k,h) \sum_{t \in h} PMAX_i^t < IMAX, \tag{3.27}$$

where $\Phi(i,k,h)$, as above, is defined by (3.25) and IT, C, and $IMAX$ as in (3.17).

3.4.1.3 Fuel Level Approximation

To enforce the bounds on refuel amounts, we implement the following constraints based on (3.2)

$$r(i,k) \geq \sigma(i,k) RMIN_{i,k}, \tag{3.28}$$

$$r(i,k) \leq \sigma(i,k) RMAX_{i,k}. \tag{3.29}$$

To estimate the fuel level before and after each outage, we need to introduce several variables, which are visualized in Figure 3.5.

Recall that XI_i is the initial fuel level at plant i. Let $\beta(i,h)$ denote the accumulated fuel usage for type 2 plant i in weeks 0 through $h-1$ of the planning horizon, assuming production at maximum capacity

$$\beta(i,h) = D \sum_{h'=0}^{h-1} \sum_{t \in h'} PMAX_i^t. \tag{3.30}$$

The $\beta(i,h)$ values are used to estimate the fuel usage during the production period preceding outage k for type 2 plant i which we denote by $FU(i,k)$. For $k = 0$ we have

$$FU(i,0) = \sigma(i,0)\beta(i, ha(i,0)), \tag{3.31}$$

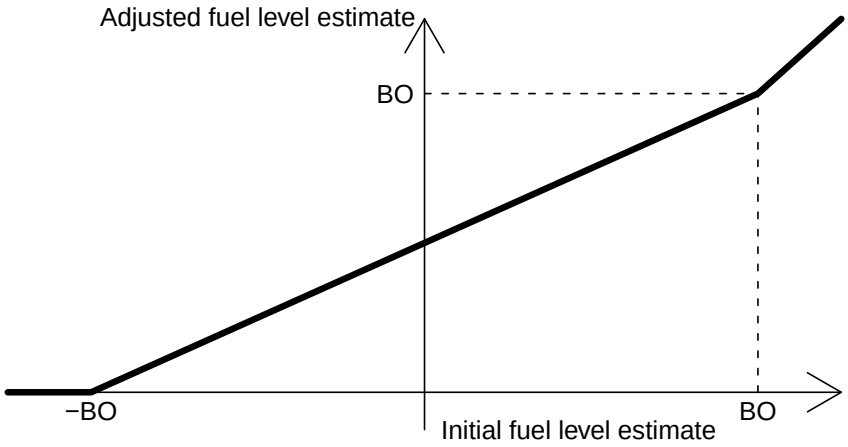

Figure 3.5: Temporal overview of notation for approximated fuel usage (FU) and fuel levels (XI, FI, FB, FA) for a type 2 plant.

and for $k > 0$ we have

$$FU(i,k) = \\ \sigma(i,k)(\beta(i,ha(i,k)) - \beta(i,ha(i,k-1) + DA_{i(k-1)})). \tag{3.32}$$

The definitions of the following three variables are mutually recursive, since they depend on values for the previous outage. An initial estimate $FI(i,k)$ of the fuel level at type 2 plant i at the time of outage k, which is subsequently adjusted to a better estimate $FB(i,k)$, is obtained as follows

$$FI(i,k) = \begin{cases} XI_i - FU(i,k), & \text{if } k = 0 \\ FA(i,k-1) - FU(i,k), & \text{if } k > 0, \end{cases} \tag{3.33}$$

where $FA(i,k-1)$ denotes the estimated fuel level after outage $k-1$ for type 2 plant i and is computed by (3.34) which is derived from (3.4)

$$FA(i,k) = Q_{i,k}FB(i,k) + r(i,k) + Q'_{i,k}. \tag{3.34}$$

Finally, we adjust the initial estimate for fuel level before an outage. The problem with $FI(i,k)$ is that the declining power profile in constraint (3.10) is ignored. This will often underestimate the actual fuel level because the plant usually produces at less than $PMAX$ at the end of the production campaign and consequently uses less fuel. Experiments show that this often leads to situations where no feasible production plan can be found for a given schedule. Thus, in order to take the declining power profile into account, we adjust $FI(i,k)$ if it is low enough that the power profile is activated in the end of the production campaign. More precisely, if $FI(i,k) < BO_{i,k}$, we assume that plant i would have run out of fuel when $FI(i,k) = -BO_{i,k}$ and make a linear interpolation. Note that the decision to choose $-BO_{i,k}$ as the cut-off point is heuristic. The adjusted fuel level estimate $FB(i,k)$ is computed

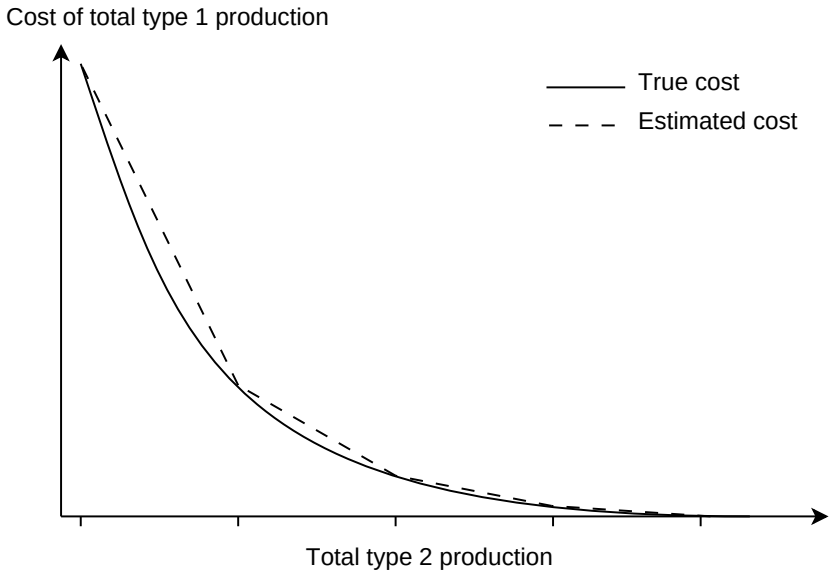

Figure 3.6: The piecewise linear function mapping an initial fuel level estimate $FI(i,k)$ to an adjusted estimate $FB(i,k)$ that takes the power profile into account. The adjusted estimate is never negative, and for any $FI(i,k) \geq BO$, the function works like the identity function.

by

$$FB(i,k) = \begin{cases} 0, & \text{if } FI(i,k) \leq -BO_{i,k} \\ FI(i,k), & \text{if } FI(i,k) \geq BO_{i,k} \\ FI(i,k) + \frac{1}{2}(BO_{i,k} - FI(i,k)), & \text{otherwise.} \end{cases} \tag{3.35}$$

This relationship between $FB(i,k)$ and $FI(i,k)$ is shown in Figure 3.6. This approximation implies that a feasible solution might be infeasible in the CP model and vice versa, but in practice the approximated constraints leads to maintenance schedules that are feasible in the sense that there exists a feasible production plan for all scenarios.

3.4.1.4 Surrogate Objective Function

The scheduling problem is primarily concerned with minimizing the use of type 1 plants, which is equivalent to maximizing the amount of online type 2 capacity. Therefore, the surrogate objective function measures average offline type 2 capacity over time. Minimization of this objective function ensures good decisions on the number of outages scheduled. Note that if too few outages are scheduled the plants run out of fuel, hence we also include this in the objective function.

Let α_i denote the average maximal fuel usage per week for type 2 plant i

$$\alpha_i = \frac{\sum_{t=0}^{T-1} PMAX_i^t}{H} D. \tag{3.36}$$

Furthermore, let the auxiliary decision variable k'_i denote the index of the last scheduled outage for type 2 plant i, i.e.

$$k'_i = \sum_{k=0}^{K-1} \sigma(i,k) - 1. \tag{3.37}$$

Then, the surrogate objective function to be minimized is

$$\begin{aligned}\sum_{i=0}^{I-1} \alpha_i \Bigg(& \Big[ha(i,0) - \frac{XI_i}{\alpha_i}\Big]^+ + \sum_{k=1}^{K-1} \Big[\sigma(i,k) \\ & \Big(ha(i,k) - ha(i,k-1) - DA_{i(k-1)} - \frac{FA(i,k-1)}{\alpha_i}\Big)^+\Big] \\ & + \Big[H - ha(i,k'_i) - DA_{ik'_i} - \frac{FA(i,k'_i)}{\alpha_i}\Big]^+\Bigg).\end{aligned} \tag{3.38}$$

The first term is the estimated offline capacity before each plant's first outage, the second term is estimated offline capacity before each of the plants' subsequent outages, and the third term is the estimated offline capacity after each plant's last scheduled outage.

3.4.1.5 Search Strategy

A CP solver finds a solution to an optimization problem by searching a tree which is pruned by applying *constraint propagation* and *branch and bound* strategies. Two decisions are crucial for making this pruning effective, namely *variable selection* (choosing the next variable to branch on) and *value selection* (choosing a value for the chosen variable). Variable and value selection strategies are generally chosen according to the *fail-fast* principle, which says that if no feasible solution exists in the current subtree, then the search should determine this as early as possible. Furthermore, finding a good solution early in the search is desirable because it improves the efficiency of branch and bound pruning.

Our variable selection strategy attempts to make consecutive decisions in the search tree that affect outages which are likely to be related by constraints; this often corresponds to outages that are scheduled close to each other. This is achieved by branching on variables grouped by outage number in the following way. A random permutation π of type 2 plants is constructed. Let $\pi(\ell)$ denote the ℓth plant in this permutation. We examine all cycles $k \in \{0, \ldots, K-1\}$, and for each k examine all type 2 plants $\pi(\ell) \in \{0, \ldots, I-1\}$. For each outage $(\pi(\ell), k)$ the variables are fixed in the following order and with the specified value selection strategy:

1. Determine whether $(\pi(\ell), k)$ is scheduled or not, i.e., whether $\sigma(\pi(\ell), k) = 1$ or $\sigma(\pi(\ell), k) = 0$. The $\sigma(\pi(\ell), k) = 1$ branch is considered first, since this leads to more scheduled outages.

2. In the $\sigma(\pi(\ell), k) = 1$ branch, determine the outage starting week $ha(\pi(\ell), k)$. The earliest possible week is considered first, since this leaves more room for subsequent outages for plant $\pi(\ell)$.

3. Determine the refuel amount $r(\pi(\ell), k)$. The maximal amount is considered first, since this leads to more type 2 capacity.

Preliminary experiments showed that this branching strategy works well. The CP solver is given at least 10 minutes to find a solution, if a solution is found the search is stopped, and if not it is continued until a feasible solution is found. Details about the CP solver software are given in Section 3.8.2.

3.4.2 Greedy Production Level Planning

We set the production levels $p(i, t, s)$ and refuelling amounts $r(i, k)$ of a feasible maintenance schedule returned by the CP solver by means of a greedy algorithm which we call *production planner*. Note that the refuelling amounts from the CP maintenance schedule are not used in the phases after CP.

Only constraint (3.5) concerning demand binds production across different type 2 plants. The production planner ignores the demand and therefore all computations can be done for each type 2 plant independently. Ignoring the demand may lead to overproduction which is fixed by modulation in a final phase, described in Section 3.7.

The algorithm starts with the first time step and goes through all time steps in the planning horizon. It uses the initial fuel level to produce at maximum capacity until no more fuel remains or the next outage is encountered. If a plant runs out of fuel in some production campaign, it cannot produce in the rest of the production campaign.

We use the production planner in two settings:

1. It is used on an initial maintenance schedule from the CP solver, and in this case the production planner initially sets refuel amounts to the minimum allowed amount $RMIN_{i,k}$ for every outage.

2. It is used repeatedly in the SLS, where the current refuel amounts are updated. The production planner first tries to achieve feasibility by reducing refuel amounts, as described in Section 3.4.2.1, and then to increase the refuelling amounts, as described in Section 3.4.2.2.

3.4.2.1 Reducing Refuel Amounts

Infeasibility with respect to fuel levels can occur if constraint (3.12) is violated because of a too high fuel level before an outage, or constraint (3.13) is violated because of a too high fuel level after an outage. If one of these situations is encountered, the production planner backtracks to the previous outage and reduces the amount of refuelling done there — this change is subject to constraint (3.2) on refuelling amounts. This is done recursively, and if the backtracking reaches the start of the planning horizon without resolving the problem, the planner declares the maintenance schedule infeasible. If this happens and the production planner is called from the local search, the infeasible neighbour is simply skipped. It has never happened

to the initial solution from the CP solver, but in such a case the problem could be returned to the CP to attempt to create a new solution.

3.4.2.2 Increasing Refuel Amounts

After having decreased refuelling amounts wherever necessary to the point where constraints (3.12) and (3.13) on fuel levels are satisfied, the production planner tries to increase the refuel amounts as much as possible in order to maximize type 2 production. This is done for each production campaign k in turn, starting with the campaign after the last scheduled outage and proceeding backwards. If plant i is able to produce at $PMAX_i^t$ for all $t \in ec(i, k'')$ where $k'' \geq k$, then there is no need to increase refuel amounts (unless the value of type 2 fuel at the end of the planning horizon exceeds its cost at the time of outage k; we ignore this possibility). Otherwise, the production planner repeatedly tries to increase the refuelling amount by $0.02(RMAX_{i,k} - r(i,k))$ until either production is maximized or the maximal allowed refuel amount is reached.

3.5 Improvement by Stochastic Local Search

When evaluating the quality of the initial maintenance schedule from the CP we see that it is of relatively low quality. This is the case since the strategy used in the CP model is to place outages mainly to ensure a feasible solution and to a lesser extent to minimize production costs. This leaves room for improving the temporal placement of outages.

Moving outages can reduce the total cost of production in two ways. First, it can increase the amount of electricity produced by type 2 plants and thereby decrease the production of the more expensive type 1 plants. Second, it can move outages to a time period where the type 1 production costs are lower.

The SLS is done in a typical simulated annealing setting, see [Kirkpatrick 1983]. The basic move in the local search is to choose a random outage and shift it a few weeks forward or backward. After each move the scheduling and resource constraints (3.1) and (3.14) to (3.17) are checked for feasibility. If the constraints are satisfied, the production planner calculates updated refueling amounts and production levels in order to check feasibility with respect to fuel levels. If a move is feasible, it is candidate for being accepted by the simulated annealing depending on the current temperature and the change in the objective function.

A limitation of the SLS procedure is that the number of outages is not changed. The effect of this is that the CP model is trusted with deciding the number of outages for each power plant.

We have tried to let the local search add and remove outages but found that this operation introduces efficiency problems. Adding or removing outages leads to changes in the whole solution, meaning that the local search spends a lot of time re-adjusting the placement of all the other outages. This gives a very long evaluation

time (several hundred iterations) for adding or removing an outage, and thus hinders an effective local search.

In the next sections, we describe the SLS approach in detail.

3.5.1 Neighbourhood

Formally, given a schedule for outages, $ha(i,k)$, $0 \leq i < I, 0 \leq k < K$, a neighbouring scheduling solution ha' is obtained by applying the move (i', k', m)

$$ha'(i,k) = \begin{cases} m, & \text{if } i = i' \text{ and } k = k' \\ ha(i,k), & \text{otherwise.} \end{cases} \tag{3.39}$$

The value m is chosen uniformly random in the interval $\left[TO_{i',k'}, TA_{i',k'}\right]$, so only neighbouring schedules that satisfy constraint (3.1) are considered. A move (i', k', m) thus corresponds to selecting outage k' of plant i' and moving it to start in week m.

The size of the neighbourhood is bounded from above by $I \cdot K \cdot H$, but the bounds in (3.1) reduce the number of neighbours significantly. The length of the interval $[TO_{i,k}, TA_{i,k}]$ is usually between 20 and 30 weeks on average (including outages where $TO_{i,k}$ or $TA_{i,k}$ are undefined, in which case the interval consists of all weeks before $TA_{i,k}$ or all weeks after $TO_{i,k}$). In two instances where very few outages have constraints of type (3.1), the average interval is around 150 weeks. The size of the neighbourhood is reduced by only considering moves (i,k,m) where the outage is moved less than n weeks forward or backward, i.e.,

$$|ha_{i,k} - m| < n. \tag{3.40}$$

Preliminary experiments have shown that a good value for n is 20, this is used for all instances.

The feasibility of a neighbour can be checked effectively because each outage is involved in a relatively low number of constraints. It is straightforward to precompute a data structure that maps an outage to a list of the scheduling constraints involving the outage. With this list, feasibility of a neighbour can be checked efficiently. If the feasibility check detects a violated constraint, the evaluation is terminated immediately and the move is discarded. This implies that the SLS never moves to an infeasible maintenance schedule.

3.5.2 Delta Evaluation and Acceptance Criteria

From the previous section we have that scheduling feasibility can be checked efficiently. Calculating the change in solution cost and checking for refuelling feasibility is however more involved. For a type 2 power plant that had an outage moved it is necessary to re-plan the production for all time steps. Only a single artificial scenario of production levels for each type 2 plant is maintained, meaning that we only recalculate production levels and implied fuel levels for one scenario. This is done both to ensure that we still have a feasible production plan and to estimate

the change in the objective function. The new production plan is calculated by the production planner described in Section 3.4.2. If the production planning is unable to find a feasible production plan, the move is discarded.

The change in objective consists of three parts: the change in cost of type 2 refuelling Δ_{refuel}, the change in cost of type 1 production Δ_{type1}, and the change in value of remaining fuel at the end of the planning horizon $\Delta_{remainder}$. The total change Δ is then

$$\Delta = \Delta_{refuel} + \Delta_{type1} - \Delta_{remainder}. \tag{3.41}$$

We now describe how to compute each of these three contributions.

When the greedy production planner has calculated new production levels for the single scenario and updated the refuelling amounts, we get

$$\Delta_{refuel} = \sum_{k=0}^{K-1} C_{i,k}(r'(i,k) - r(i,k)), \tag{3.42}$$

where $r'(i,k)$ is the refuel amount in the neighbouring solution and $C_{i,k}$ the fuel cost in outage i,k.

The amount of fuel remaining at the end of the planning horizon can vary from scenario to scenario. Here we estimate the remaining amount assuming that every type 2 plant has full production in all time steps, meaning that the estimate is a lower bound on the actual amount of remaining fuel. This gives

$$\Delta_{remainder} = C_i(x'(i,T,s^*) - x(i,T,s^*)), \tag{3.43}$$

where $x'(i,T,s)$ is the fuel level in the neighbouring solution, C_i the value of remaining fuel, and s^* is the artificial full production scenario.

Calculating Δ_{type1}, the change in total cost of type 1 production, is more complicated and described in Section 3.5.3.

When the objective function change Δ for a feasible move has been calculated, it is considered by the simulated annealing. A neighbour is accepted with probability

$$p = \min\left(1, \exp\left(-\frac{\Delta}{\tau}\right)\right), \tag{3.44}$$

where τ is the current temperature.

The initial temperature is dynamically set such that approximately half of the considered moves are accepted at the beginning of the search. The cooling scheme is geometric with cooling ratio c and is applied every $n_{plateau}$ iterations. When m_{idle} consecutive iterations have been considered without any move being accepted, a restart is performed. When the search is restarted, the current temperature is set to $k_{restart}$ times the starting temperature of the previous annealing run.

At the end of the time allocated to the SLS, a first-fit descent is performed to ensure that the search reaches a local optimum.

3.5.3 Estimation of Type 1 Production Cost

First we explain how the exact change in type 1 production cost is calculated. Then we introduce an effective way to approximate this change, since the exact computation is too expensive to be suited for use in local search.

To help calculating Δ_{type1} efficiently, we maintain a list of the total type 2 production in every time step. We look at the type 2 plant which had an outage moved and use the change in production in each time step, to update the list of total type 2 production. In scenarios where the demand is higher than the total type 2 production, the difference must be covered by type 1 plants, which have a fixed cost per unit of power produced.

The type 1 plants can be preordered by increasing production costs, and thus the exact change in total type 1 cost, given a change in total type 2 production, can be computed in $O(ln(J))$ time for a single time step and scenario, as explained below.

Assume that the previous total type 2 production in time step t was $p_{T2}(t)$ and after a move to a neighbouring maintenance schedule, the total type 2 production is adjusted to $p'_{T2}(t)$. To calculate the cost of the residual type 1 production for a scenario s we examine the ordered list of type one power plants. While the residual demand $DEM^{t,s} - p'_{T2}(t) - p'_{T1}(t)$, where $p'_{T1}(t)$ is the current type 1 production, is positive we increase the production of the type 1 plant with the lowest production cost and available production capacity. At some point the residual demand will be satisfied by the extra type 1 production.

The above procedure can be seen as a piecewise linear function for the cost in each scenario, mapping type 2 production to the cost of covering the residual demand using type 1 plants. Finding the linear piece of this function that corresponds to a given total type 2 production can be done by binary search, which reduces the computational complexity of finding the total type 1 production cost for a single time step to $O(ln(J))$.

The above procedure gives the type 1 production cost for a single time step and a single scenario. Summing this exact change over all time steps and scenarios gives a total complexity of $O(T \cdot S \cdot ln(J))$ which is too slow for local search.

Hence we approximate the change in cost. The approximation removes the need to consider all scenarios and the binary search through the set of type 1 plants but not the need to consider all time steps, making the complexity of the approximation $O(T)$.

To estimate the change in type 1 cost for a single time step, we precompute a single piecewise linear function which maps total type 2 production to the (exact) total type 1 cost for each time step. This is done by summing all S piecewise linear functions for a single time step.

The solid line in Figure 3.7 shows an example of such a function. The curve seems smooth but is, since it is a sum of piecewise linear functions, also a piecewise linear function. As the total type 2 production increases, the need for type 1 production diminishes, and when total type 2 production reaches the maximum demand over

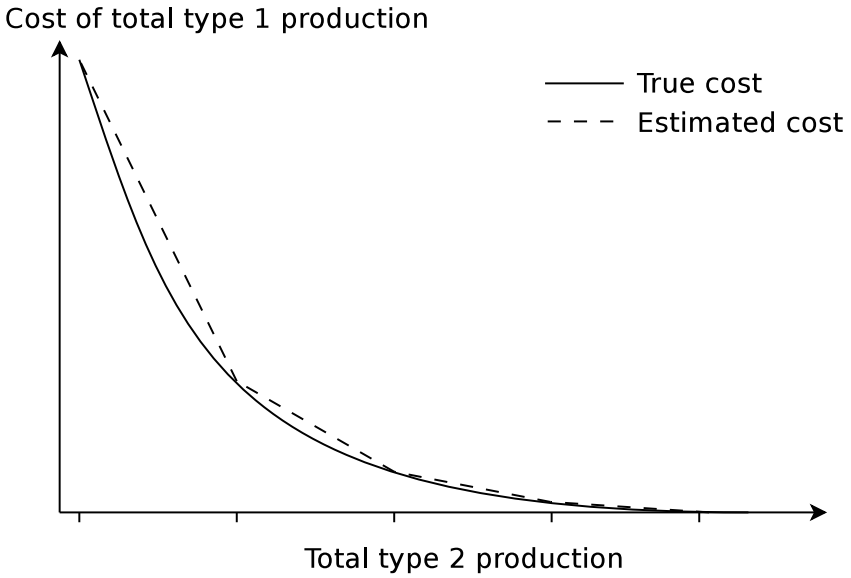

Figure 3.7: Linear approximation of the total type 1 costs for a single time step.

all scenarios, the total type 1 cost is zero.

The many breakpoints make the evaluation of function computationally expensive because it must go through the breakpoints in order to find the interval containing the current total type 2 production (a binary search can speed up this process but is still too slow). Therefore, we approximate it with another precomputed piecewise linear function, which has fewer breakpoints chosen such that they are equidistant on the x-axis. The dashed line in Figure 3.7 shows an example of this approximation. Since the actual type 1 cost is convex, the approximation is an upper bound on the actual cost.

Experiments show that this approximation is relatively good, even for a reasonably small number of breakpoints. We have chosen $3 \cdot I$ breakpoints, and the approximation error has been analysed on all instances. If absolute type 1 production demands under 100 units are ignored, the worst average deviation over an instance is $\approx 3.36 \cdot 10^{-4}$ percent per measurement. This is significantly less than the differences in cost encountered during local search. When evaluating the very small type 1 production demands we ignored above, the relative approximation error is significant, but this does not hinder the SLS, since these moves only give correspondingly small changes in the objective value.

Using the precomputed approximation we can perform a constant time evaluation of the total type 1 cost for a given total type 2 production in a single time step. No search for the stored breakpoint closest to the given total type 2 production is required, because the breakpoints are equidistant. If the total type 2 production falls between two stored breakpoints, we use linear interpolation to get the total type 1 cost. If the total type 2 production in time step t is P_2^{total}, the interpolation

for the type 1 production cost is done between the indices:

$$i_{low} = \left\lfloor \frac{P_2^{total}}{Int} \right\rfloor, \qquad i_{high} = \left\lceil \frac{P_2^{total}}{Int} \right\rceil, \tag{3.45}$$

where P_2^{total} is the sum of type 2 production in time step t, and Int is the length of the interval between two breakpoints.

The approximation of the change in total type 1 cost is very fast compared to the exact computation but slow when compared to the scheduling constraint violation check, since the approximation requires a computation for every time step. Note however that since we only move to feasible solutions, the approximation is only performed if all the scheduling constraints are satisfied.

3.6 Changing the Number of Outages

Changing the number of outages for a type 2 power plant can be done, either by inserting an extra outage after the last in the power plant schedule, or by removing the last outage in the schedule. Removing the last outage might violate the demand constraint (3.15), even if this would not happen if the other outages were spread sufficiently. Likewise, adding an outage might violate the fuel level constraints (3.2) without this being the case had the outages been spread properly. Even given that no infeasibility is introduced by the operation, the contribution of the plant to the objective function is likely to increase from its previously optimized state.

Both methods have been tested without obtaining any success, it was particularly challenging to quantify the impact such an operation might have on the objective function after reoptimization.

3.6.1 Rescheduling a Power Plant

To alleviate the problems with simply removing and inserting an outage, we have implemented a heuristic removing all outages for a type 2 power plant, and reschedule the plants outages.

The reinsertion algorithm is based on the observation that when inserting an extra outage for a plant, the fuel constraints can be expressed as a set of weeks for which they can be fulfilled. To maintain this time window, the maximum and minimum attainable amount of fuel after each outage is stored during reinsertion. Given the rest of the schedule, the scheduling, resource and demand constraints each impose constraints on which of the weeks in the time window defined by the refuelling constraints, are valid candidates for inserting the next outage.

To determine candidate weeks for scheduling the next outage of the plant whose outages we are rescheduling, we iterate from the week the plant runs out of fuel to the first week an outage can be scheduled under refuelling constraints (3.2), and select the first feasible as shown in Algorithm 5. If no feasible week is found within that interval, we iterate through the weeks in which the plant is out of fuel in increasing order. After an outage is scheduled we attempt to schedule the next outage using

the same algorithm, terminating when either the plant is fuelled the entire schedule, or the next outage would lie outside the scheduling horizon. The iteration order is chosen to maximize the production of power from the type 2 plant. This is too far from the real objective function for this strategy to be used in a destruct/reconstruct heuristic except in the initial stages of the local search.

To avoid the potentially exponential number of backtracks in Algorithm 5, we impose a limit on these. The limit was never encountered in practice, so the algorithm never had to restore the original schedule. It is worth noting that a feasible insertion will always exist, namely the one utilized before the rescheduling.

3.6.1.1 Determining Feasible Weeks for Insertion

Each scheduling, demand and resource constraint can be modelled using resources that are required to be present in given weeks relative to the start of the outage, and consumed in a potentially different set of weeks relative to the start of the outage:

Scheduling constraints (3.14) are represented using a binary resource. The resource is required to be present at the starting week of the outage, and is consumed for an interval around the start week determined by S_e and S'_e. No resource is ever used outside the interval of weeks where the constraint is defined, and if an outage is scheduled outside the interval, its resource usage is not registered.

Resource constraints (3.16) are modelled directly as using one unit of the resource for the entire outage. N units are initially available at any given week.

Demand constraints (3.15) uses the resource approximated as $max_{i \in h} PMAX_i^t$ if the outage is scheduled in week h and the available pool is $IMAX$. The approximation adopted by using maximum values was shown to have no noticeable effect as $PMAX_i^t$ varies little over any given week.

We maintain a list of the amount of each resource available at any given week, and decrement these values appropriately when scheduling a new outage. The function $feasible(i, k, h)$ then checks each constraint for violation in constant time when determining if plant i's outage k can be scheduled in week h.

3.7 Modulation

In this last phase of the solution process, we ensure that there is no overproduction in any scenario. We also calculate production levels for all scenarios. This gives the final solution to the energy management problem. In this phase, we do not move any outages but do adjust refuelling amounts for some of the outages.

From the SLS we have a maintenance schedule where the cost of covering demand by type 1 power plants is minimized. This may lead to solutions with overproduction

Data: A function $feasible(i, k, h)$ mapping outage weeks to the feasibility of scheduling outage k for type 2 plant i in week h.
A plant i and an outage k to reschedule.

if $k = 0$ **then** `// We reschedule the first outage`
  Remove the plant i's outages from the schedule;
  Update $feasible(i, k, h)$;
end
if *plant i has the required outages scheduled, and has fuel or an outage at the week H-1* **then**
  return *true*;
end
$h_{nofuel} \leftarrow$ week plant i runs out of fuel ;
$h_{first} \leftarrow$ week plant i is low enough fuel for an outage ;
for $h_{schedule} \leftarrow h_{nofuel}$ **to** h_{first} **do**
  if $feasible(i, k, h_{schedule})$ **then**
    $h(i, k) \leftarrow h_{schedule}$;
    Update $feasible(i, k, h)$;
    if $schedule(i, k + 1)$ **then**
      return *true*;
    end
    $h(i, k) \leftarrow -1$;
    Update $feasible(i, k, h)$;
  end
end
for $h_{schedule} \leftarrow h_{nofuel} + 1$ **to** $H - 1$ **do**
  if $feasible(i, k, h_{schedule})$ **then**
    $h(i, k) \leftarrow h_{schedule}$;
    Update $feasible(i, k, h)$;
    if $schedule(i, k + 1)$ **then**
      return *true*;
    end
    $h(i, k) \leftarrow -1$;
    Update $feasible(i, k, h)$;
  end
end
return *false*;

Algorithm 5: schedule(i,k)

at type 2 plants due to the variations in demand across scenarios. Formally, we have in some time step t in some scenario s

$$\sum_{i=0}^{I-1} p(i,t,s) > DEM^{t,s}.$$

Such overproduction can be eliminated in two ways: modulation can be used to decrease the power output of a type 2 plant in the affected time step, or refuelling can be decreased such that a type 2 plant runs out of fuel before that time step. Making a power plant run out of fuel is not an attractive option because it eliminates the rest of its production for the current campaign across all scenarios, since refuelling amounts are shared among all scenarios. It will also limit the plant's fuel level in the next production campaign. Therefore this method is used as a last resort, i.e., modulation is used whenever possible — and it is capable of eliminating overproduction in all the examined instances.

There are three constraints on the amount of modulation that can be performed on a single power plant: constraint (3.8) restricts the amount of modulation that can be performed in the current campaign, and constraints (3.12) and (3.13) enforce an upper limit on the fuel level before and after refuelling. The latter must be handled by adjusting the refuelling of the power plant, but this will affect all scenarios.

Since refuelling amounts for a plant are shared among all scenarios, we use a two-step procedure to eliminate overproduction. First, we ensure that there exists a refuelling scheme that is feasible for all scenarios using the minimum demand scenario defined below. Second, we try to adjust modulation for each individual scenario to reduce the need for expensive type 1 production.

We define a minimum demand scenario to be $DEM^t_{min} = \min_s(DEM^{t,s})$ for all t. By using this scenario we can ensure feasibility of all scenarios, since the only way the demand influences feasibility is by making modulation necessary to get the type 2 production low enough to match demand in constraint (3.5). As type 2 power plants according to [Porcheron 2010] deliver 87% of the combined power on average, such situations arise sparingly, and the minimum demand scenario can be modulated to feasibility in all instances used in the competition.

3.7.1 Modulation and Refuelling for the Minimum Demand Scenario

A modulation and refuelling scheme is created for the minimum demand scenario, thus ensuring that the refuelling is feasible for all scenarios. This is done by examining the time steps in increasing order. When a time step with overproduction is detected, a target plant i is selected among the type 2 plants. Plant i has its production lowered until there is no more overproduction or as much as feasible while respecting the remaining modulation capacity in that campaign, see equation (3.8). This reduces the fuel consumption, meaning that the fuel level constraints must be checked. If necessary, we repair the refuelling amounts at plant i using the greedy production planner. If this fails, the modulation on plant i is undone, and another

plant is selected. We continue selecting a new plant and reducing its production until we have eliminated any overproduction in the current time step. Note that since we consider the minimum demand scenario, we are guaranteed that there is no overproduction in any scenario.

Since the cost of modulating each type 2 plant is the same, modulation can be seen as a resource that expires when the production campaign ends. Thus, the target plant selection strategy iterates through plants in non-descending order of their current production campaign's end date.

A refuel plan may be infeasible for the minimum demand scenario but still be feasible for all scenarios. Our method is unable to cope with this situation and will therefore declare such a schedule infeasible, but this has never occurred in any instance used in the competition.

3.7.2 Modulation per Scenario

Having determined modulation for the minimum demand scenario, we fix refuel amounts and apply the modulation algorithm from the previous subsection on each scenario. Modulating without consecutive adjustment of the refuelling amounts can yield infeasible fuels levels, meaning that modulation cannot always be done for a plant. In this case the algorithm moves on to the next type 2 plant.

On some instances this step improves the objective value by more than 1%. This improvement was done after the competition.

3.8 Computational Analysis and Results

In this section we describe the problem instances used for computational tests, implementation details, how much time is spent in different components of the heuristic, tuning of the parameters in the simulated annealing algorithm, and finally we report the numerical results.

3.8.1 Problem Instances

We have tested our algorithm on ten real-life instances supplied by Électricité de France. Table 3.1 shows various figures as well as the best known objective value for each of the instances, which are taken from the ROADEF website [Porcheron 2010]. Note that the B and X instances are pairwise very similar, e.g. $B6$ and $X11$, $B7$ and $X12$ and so on. This similarity is due to the fact that they are based on the same data, but have different demand profiles.

3.8.2 Implementation Details

The algorithms are implemented in Java, and the scheduling problem is solved using the Gecode CP solver version 3.3.1 [Schulte 2010]. The version of our program that was submitted for the qualifying phase used ILOG's CP Optimizer version 2.3 instead of Gecode, but the former model was unable to solve the larger problem

Instance	File size	T	Weeks	S	J	I	Best solution
B6	140	5 817	277	50	25	50	83 424 716 217
B7	144	5 565	265	50	27	48	81 174 243 138
B8	262	5 817	277	121	19	56	81 926 206 073
B9	262	5 817	277	121	19	56	81 750 858 197
B10	252	5 565	265	121	19	56	77 767 024 999
X11	140	5 817	277	50	25	50	79 116 772 289
X12	143	5 523	263	50	27	48	77 589 910 940
X13	262	5 817	277	121	19	56	76 449 207 715
X14	262	5 817	277	121	19	56	76 172 998 633
X15	250	5 523	263	121	19	56	75 101 398 439

Table 3.1: Overview of the ten instances showing for each instance file size in megabytes, number of time steps (T), number of weeks, number of scenarios (S), number of type 1 plants (J), number of type 2 plants (I), and the objective value of the best known solution.

instances used in the final round.[4] Moreover, the Gecode solver allows the user control over the applied branching strategy. On the basis of preliminary observations, we decided to stop the CP solver after ten minutes and then return the best solution found. If no feasible solution has been found after ten minutes, we let the solver run until the first feasible solution is found.

The strategy described in Section 3.4.1 finds a feasible schedule solution in less than two minutes for all instances, as seen in Table 3.2. The table also shows the minimum time needed to construct a solution, using the initial maintenance schedule from the CP and applying the modulation phase but without SLS.

	B6	B7	B8	B9	B10
CP (s)	84	67	11	22	28
Mod (s)	7	7	48	43	43
Total (s)	167	153	256	237	263
% gap	12.17	6.73	16.26	19.79	9.80

	X11	X12	X13	X14	X15
CP (s)	13	8	13	9	15
Mod (s)	8	10	70	71	49
Total (s)	108	94	315	366	242
% gap	9.09	5.80	14.97	15.52	7.37

Table 3.2: The time in seconds needed to produce the first feasible scheduling solution (CP), modulating it (Mod) and total time including reading from and writing to disk (Total). The last row is the percent-wise gap from best known objective value.

[4]Note that the model was at fault and not necessarily the solver. Gecode was chosen due to the authors familiarity with the framework.

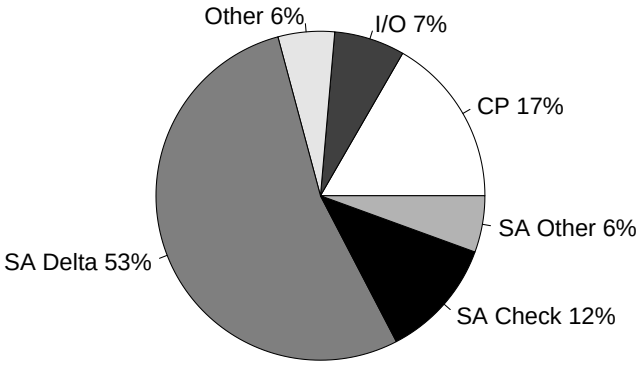

Figure 3.8: How one hour of wall clock time is spent when solving instance B10.

In the competition a machine with a 2.50Ghz Intel Xeon processor was used, while our corrected results were created using a 2.13GHz Intel Xeon processor. In both cases 8GB of main memory were available.

3.8.3 Time Allocation

Preliminary tests indicated that ten minutes for the CP solver and the remaining 50 minutes for local search and other tasks is a reasonable distribution of the one hour available. Letting the CP solver continue after having found a feasible solution results in more scheduled outages for each type 2 plant, which is important since the subsequent local search does not change the number of scheduled outages. However letting the CP solver run longer than the allocated ten minutes is not the best time utilization because the correlation between the surrogate and real objective function is not that strong. The current division of time between CP and SLS is thus our best attempt at creating initial solutions with a good number of outages without using too much time.

The pie chart in Figure 3.8 shows how much time is usually spent in different parts of the program. Not surprisingly, most time is spent on the simulated annealing algorithm, whose delta evaluation alone accounts for more than half of the total running time. This is due to the fact that evaluation of a neighbour requires replanning of production levels which is very time consuming, even when applying the approximation described in Section 3.5.2. Somewhat unusual is the 7% of the total running time spent on reading an instance and writing a solution to the hard disk, which is caused by the very large instance and solution files. The latter takes up to 950 megabytes of hard disk space.

3.8.4 Tuning the Stochastic Local Search

To determine a set of parameters that perform well for all problem instances we compared a number of different parameter settings. All the following combinations of parameters were run for all ten instances in sets B and X and for ten different

random seeds. In order to reduce the number of configurations, we decided after preliminary testing to fix the number of moves at each temperature to 100 and that the initial temperature after a restart should be twice the initial temperature of the previous run.

Cooling ratio $c = \{0.95, 0.96, 0.97, 0.975, 0.98, 0.985, 0.99, 0.995\}$

Start acceptance ratio = $\{0.25, 0.5, 0.75\}$

Stopping criteria $m_{idle} = \{25, 50, 75, 100, 125, 150, 175, 200, 300, 500, 800\}$

Number of moves per temperature plateau $n_{plateau} = 100$

Reheat constant $k_{restart} = 2$

The best average solution quality is obtained by setting the cooling ratio to 0.995, the start acceptance ratio to 0.5 and the stopping criteria to 100. This setting also has a reasonably low variance of results compared to other settings.

3.8.5 Results

Team	B6	B7	B8	B9	B10	X11	X12	X13	X14	X15	Score
J06 (corrected)	**2.45**	**0.81**	**1.04**	**1.23**	**1.66**	**1.30**	**1.00**	**1.44**	**1.70**	**0.43**	**13.04**
S21	4.14	3.09	6.30	6.29	3.70	3.13	1.58	5.41	4.29	1.28	39.21
J08	3.85	6.14	16.94	23.83	9.19	2.85	2.66	2.48	5.15	4.03	77.12
S14	11.55	9.48	13.13	25.82	12.66	10.37	11.13	12.85	18.03	14.88	139.89
S23MT	1.54	0.82	1.28	2.29	1.24	TO	0.78	INF	TO	**0**	202.36
S17	19.44	25.17	39.71	56.68	34.47	16.82	23.14	TO	20.87	20.53	297.19
S24	**0**	0.14	**0**	1.06	**0**	**0**	0.06	**0**	**0**	INF	299.96
J06	**2.50**	**0.96**	**1.94**	**2.12**	**3.11**	**1.46**	**1.11**	**2.91**	**INF**	**INF**	**407.5**
S04	7.68	6.41	15.64	26.80	8.58	TO	6.50	INF	14.01	TO	486.04
S22	0.59	0.06	0.18	**0**	0.33	INF	0.01	TO	ERR	ERR	494.29
S08	INF	17.13	24.90	24.68	40.22	10.20	6.50	20.18	13.85	INF	495.24
J05	3.34	3.43	339.67	162.91	22.43	3.79	1.52	3.35	14.35	19.41	574.20
S23	1.54	0.82	1.33	2.28	1.30	TO	TO	INF	TO	TO	624.03
S16	3.44	INF	87.05	47.31	7.15	INF	2.59	TO	TO	INF	773.51
S10	7.39	66.42	3685.91	4779.61	93.76	17.94	15.98	TO	46.34	48.21	8801.92
S10MT	7.49	66.42	3685.91	4779.61	93.76	30.69	61.82	TO	46.34	149.35	8961.75
J16	11.10	12.66	TO	1845.78	12.20	7.70	8.09	TO	12.81	8.11	9330.63
S11	9.47	6.26	1902.45	MEM	INF	6.52	7.41	MEM	11.42	INF	12029.32
S22MT	0.60	**0**	INF	INF	INF	INF	**0**	TO	ERR	ERR	17612.27
S25	CRA	CRA	CRA	CRA	CRA	CRA	CRA	CRA	CRA	CRA	19907.04

Table 3.3: Percentage wise deviation from best known solution for all teams participating in the competition as well as our improved program. Our team is J06. *INF* means that the found solution is infeasible. *CRA* means that the program crashed. *MEM* means that the program ran out of memory. *ERR* means that the format of the solution is invalid. *TO* means that the program did not finish in time.

Table 3.3 shows, for all teams participating in the competition, the percentage wise deviation from best known solution for each instance. The last column shows

the teams' official final score which determined the outcome of the competition[5] . This score is the sum of all ten percentages. If a team was unable to find a feasible solution for an instance, their score for this instance was set to twice the objective value of the worst found solution.

Our team identifier is J06 which is ranked seventh in the table. The first row in the table shows the objective values obtained by our program after fixing the bug in the modulation procedure. The bug happened when calculating the allowed modulation in a production campaign, where we did not account for the size of a time step in one special case, this yielded infeasible solutions for X14 and X15. The results from the table show that our new corrected program would have won competition.

The difference in solution quality between our corrected and uncorrected version of the algorithm is due to the use of modulation per scenario described in Section 3.7.2.

3.9 Conclusion

We have developed a solver for a large-scale real-life optimization problem using a combination of constraint programming, greedy heuristics and local search heuristics.

The problem had multiple objectives that were weighted a priori and merged into a cost function. Due to the large scale of the problem, stochasticity is handled proactively.

An initial solution to the complex scheduling problem is found by CP using approximated constraints for production levels and fuel consumption. From this first schedule we apply a stochastic local search algorithm based on a simple neighbourhood structure with a fast, but approximative evaluation of the objective function. In the third and final phase we use a greedy algorithm to remove any overproduction.

The solutions obtained are competitive when compared to those found by other teams participating in the final evaluation of the ROADEF/EURO Challenge 2010. After fixing an implementation bug, our approach is robust in the sense that it is always able to find a feasible solution, and it achieves overall the best score, ranking first in the assessment procedure of the competition.

[5]The results can been seen at http://challenge.roadef.org/2010/en/results.php

Chapter 4

A Framework for Dynamic Rescheduling

Contents

4.1 Contribution of the Author

This chapter is based on a paper in progress [Larsen 2012], detailing joint work with Marco Pranzo. The framework is inspired by the alternative graph model, and its applications in real life instances. (for instance [Larsen 2008]) The concept has been fleshed out through repeated discussions.

All implementations of the framework as well as all test done with the framework were done by the author.

4.2 Introduction

Scheduling problems are among the hardest and the most studied problems in operations research. However, as observed by many authors [Maccarthy 1993, McKay 1999], there is still a gap between theory and practice, in fact most formulations of scheduling problems usually assume that data is deterministic and static over time. It is well known that both uncertainties and dynamic environments are often present in real life applications. Nevertheless these presences are often neglected since they are hard to model and to take into account algorithmically. Simplifying assumptions are done to assume all information deterministic, static over time and known in advance. In some contexts, such as theoretical scheduling problems or almost deterministic applications, these assumptions may be reasonable since they clearly lead to simplified and more tractable models. This is not always the case, in fact, especially when a dynamic real life problem is tackled, the deterministic and static assumptions may be not be acceptable. In order to distinguish the two cases, a careful validation analysis on the problem is often required to assess whether simplified models are reasonable or if they induce an oversimplification. If these assumptions should lead to an unacceptable divergence between practical implementation of the plan and the expected result from the model, then the results of the optimization process are not to be trusted anymore. As a consequence, a modification of the proposed solution algorithm to incorporate more details of the problem may be required.

When the problem is assumed to be non deterministic several approaches have been introduced in the literature, ranging from extension of the optimization solvers to simulation based approaches. The problem is to formulate a plan in which the presence of the uncertainties has been taken into account and considered during the solution process. Depending on the environment being considered, the plan must be produced before the schedule execution starts and either it cannot updated during the schedule execution, or it may be updated (possibly according to some problem dependent conditions) to take the uncertainties into account.

According to this classification at least two approaches have been proposed in the literature:

1. Approaches that try to produce the best possible plan incorporating all the available information on the uncertainties. Among these approaches we can

mention *stochastic programming* [Ruszczynski 2003] and *robust optimization* [Ben-Tal 2002] approaches. In stochastic programming the idea is trying to include the uncertainty sources directly in the formulation of the problem by generating and solving several scenarios, if necessary allowing changes in the solution over time. However allowing recurse decisions makes the problem fundamentally intractable. In robust optimization approaches the idea is to try to produce a robust solution satisfying additional constraints taking into account possible realizations of the uncertainty and does not allow the plan to change once the execution is started.

2. Approaches that try to incorporate the dynamic nature of the problem such as dynamic rescheduling approaches, in which it is possible to modify or update the plan when information about the uncertainty is revealed. In this case the problem is usually maintained deterministic and the presence of the uncertainty is addressed in a external wrapper that includes the scheduling solver. As time passes, whenever some conditions are met, the deterministic scheduling solver is invoked and a new problem corresponding to a current "snapshot" of the system is solved. This process can be iterated hundreds of times. The results are then considered as a new plan to be followed during the execution.

The advantages of simulation based approaches over stochastic or robust optimization are threefold:

Tractability Simulation based approaches allows solving larger instances within reasonable computation times allowing, if application permits, hundreds of reschedules over time.

Easiness of implementation This approach does not require the development of a new solver for the problem at hand, if the deterministic solver can be adapted to meet some simple requirements. Hence, it can be put in production faster.

Knowledge of the problem Since no new solver is developed there is no need of actually knowing something about the uncertainty, to embed it in the solver. This approach can easily be adapted to evaluate the sensibility of the system with respect to errors in the uncertainty modelling, i.e., the effects of different probability distributions and parameters.

However there are also shortfalls for this approach. In fact, since the solver does not consider uncertainties, better performances can be achieved if the solver takes the information on the uncertainties into account within the optimization process. Moreover, no theoretical results are known for this kind of approach.

In this chapter we take a dynamic rescheduling approach, in which a deterministic scheduling solver is dynamically executed to update the plan in order to adapt uncertainties as they happen. The focus of this work is to introduce a general framework to model a dynamic rescheduling problem in an unified way. The main components of the system are:

Solver The solver is in charge of actually solving at each iteration the scheduling problem and thus producing a plan that is going to be executed. The solver will typically be deterministic, but this is not required.

Simulator The simulator component loads all the information on the real life realizations of the probability functions when started, and it is responsible for simulating the unpredicted deviations and disturbances that may happen in real-time. Note that the information is not sent to the solver. If the simulator is used in a real life environment, the real life data can be supplied as it becomes available.

Triggers The triggers are checked during the run of the simulation. Whenever a trigger condition is met, a new rescheduling problem is formulated according to the information available at the moment and solved by the solver.

Instance reader/writer These two modules provide the interface between the solver and the simulator. Since the possible applications pose problem-specific constraints on e.g. frozen times, rolling horizons, type of possible rescheduling, they should be implemented for each application.

In this chapter we want to demonstrate the use of the framework by shedding light on some questions, namely when the uncertainty starts to be relevant in the production of the plan in terms of feasibility and optimality and how different rescheduling policies behaves. The answers to these questions may be problem specific, however it is important to have a single tool able to address them. We are not interested in any specific application. In fact, in our tests we will apply the framework to solve two classical scheduling problems. An "easy" single machine scheduling problem and a job shop scheduling problem. The emphasis is given to the general applicability of the method rather than to the specific application or solver used.

The chapter is organized as follows: In the next section we introduce definition and notation specific to this chapter. Section 4.4 briefly reviews the related literature and in Section 4.5 we introduce and describe the proposed framework and all its structural components in detail. Section 4.6 shows the application of the framework to two different scheduling problems. Finally conclusions and future research directions conclude the chapter.

4.3 Definitions and Notation

A scheduling problem can be classified as *static* if all the data is available at the planning stage and no new information is added to or modified in the problem during the execution of the planned schedule. On the other hand, *dynamic* scheduling refers to problems where data may change during the execution of the scheduling and some information is not available to the scheduler at the planning stage.

A problem is *deterministic* if all information is certain. In *stochastic* scheduling problems, some data may be uncertain at planning stage. Different kinds of information could be available to the scheduler such as the probability function or expected values. A *stochastic instance* has probability functions associated to each process time, and the deterministic instance resulting from sampling these distributions will be called a *scenario* or a *sample instance*.

When solving a scheduling problem, two phases can be distinguished namely, a *planning phase*, in which the scheduler has to plan a schedule to solve the instance at hand, and an *execution phase*, in which the plan is executed and, depending on the application, it may be modified or not. Usually in scheduling research the main focus is on the planning phase, i.e., in producing good plans for the execution phase. ([Ouelhadj 2009])

A *disruption/disturbance* of the plan is an unforeseen event that affects the plan typically during its execution. It may consist in the realization of uncertain event which was known in the form of probability function, or it may be a more disruptive event such as the break down of a machine, failure of operations, the arrival of some new and urgent job/order to be processed or cancelled jobs etc. When a disruption occurs, a decision may be taken if the application setting makes it possible. Namely, the scheduler has to decide whether to do nothing or reschedule. In the former case, no actions are carried out and the scheduler continues to follow the plan. While in the latter case the scheduler should build/update the previous plan to reflect changes due to the disruption.

Rescheduling is the act of modifying the offline plan in response to disruptions. The rescheduling is usually an expensive activity in terms of costs, time and/or information exchange. Depending on the actual application the reschedule may not be feasible at all or it can be allowed with continuity as a monitoring process executed along the schedule execution process. The *rescheduling frequency* states how often a reschedule process can be pursued. The *rescheduling policy* regulates when a new rescheduling process can be started. Rescheduling policies can be classified as periodic, continuous, event-driven or hybrid. The periodic policy states that a rescheduling can be started after a fixed amount of time. In continuous rescheduling the rescheduling process is carried out after every timestep or event. Hybrid rescheduling is a strategy in which rescheduling is started after a fixed time interval or in response to some events. The event-driven policy states that a reschedule can start in response to a some specified events. It can be noted that the event-driven category is the most general since it can contain all the other cases. The *reaction time* is the time between a disturbance and when the solver starts the rescheduling process. Clearly, it depends on the rescheduling frequency but there may also be a minimum reaction time caused by the physical system. Finally a *rescheduling objective* has to be specified. We distinguish between complete optimization in which the rescheduling algorithm optimizes the same objective function used in the planning phase and a partial reoptimization in which the aim of the rescheduling is to minimize a surrogate objective function. Partial rescheduling (schedule repair) occurs when the rescheduling process tries to minimize the deviation from the available

offline plan and possibly taking the "real" objective function into account. This is common is some industrial or train control applications where the system should follow a plan known in advance. An example of this will be shown in chapter 5. Observe that, the deviation with respect to the plan can be limited also by acting on the constraints of the problem, i.e., by keeping precedence relations fixed or imposing time windows on the starting/ending time of the operations. Other approaches may consider a multiobjective problem in which both the "real" objective function and the surrogate function to consider stability are simultaneously optimized. It can be expected that a complete rescheduling may yield to better performance but it may cause higher costs when being implemented since the plan could be completely changed during the execution with a short notice.

The *rolling horizon* is the length of the time period for which a schedule has to be produced. Events outside the rolling horizon are not to be considered in the rescheduling. The presence of a short rolling horizon causes smaller instances since one has to schedule only the operations within the rolling horizon but it may generate myopic schedules, while longer rolling horizon generates larger instances and potentially leads to better schedules. An acceptable trade-off between these two contrasting needs should be found based on the problem being treated.

The *frozen period* is the length of the part of the schedule that cannot be changed, and that should be maintained. Operations already in execution or operations with imminent starting times that cannot be postponed, usually are considered *locked*, i.e., in the frozen period. Technological constraints may cause different length of frozen periods depending on the actual application. Observe that, the presence of a frozen period causes a partial rescheduling, a rescheduling in which the operations to be scheduled are only a subset of all the operations.

Another time of interest is the *allowed computation time*. The allowed computation time is the maximum time allotted to the optimization algorithm to compute a schedule. It is clearly dependent on the application and it is influenced by the rolling horizon and frozen times. Some applications may require fast algorithms whereas in others settings the length of the allowed computation time for the rescheduling can be comparable to the CPU time for calculating the offline schedule.

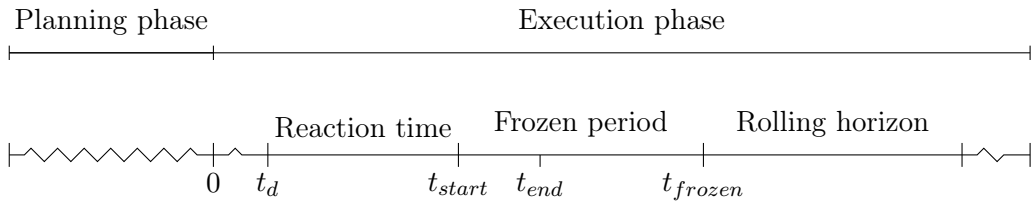

Figure 4.1: A time line for rescheduling scenarios. A disturbance occurs at time t_d, rescheduling starts at t_{start} and ends before t_{end}.

Figure 4.1 illustrates the frozen period, the rolling horizon and the allowed computation time ($t_{end} - t_{start}$). When simulating, the rescheduling framework can

reduce the complexity by considering an allowed computation time $t_{computation}$ and all operations starting before t_{frozen} locked.

4.3.1 Mathematical Notation

In this chapter we assume that the deterministic scheduling problems solution can be represented using a temporal network (i.e., a graph representation) such as [Roy 1964, Elmaghraby 1995, Mascis 2002]. The scheduling problem can be represented as a set of events/operations $\{o_0, o_1, \ldots, o_n\}$, a *minimum starting time* t_i associated to each event and a set of precedence relations among operations. Recall that a *precedence relation* (i, j) is a constraint on the starting time of operation o_j which must be greater or equal to the starting time of the predecessor o_i plus a deterministic given processing time p_{ij}.

The source and the sink operations are denoted by o_0 and o_n, and without loss of generality we assume $t_0 = 0$.

By the definitions in section 1.3.3.2, finding a schedule corresponds to assigning starting times $t_0, t_1, \ldots, t_n$ to operations $o_0, o_1, \ldots, o_n$ respectively, without causing a positive length circle in the temporal network.

Scheduling decisions are often representable as disjunctions. That is, one of the two alternative disjunctive arcs have to be selected, i.e., added to the current graph. The goal is to *minimize* a linear combination of the starting time of the operations.

Problem 4.3.1.

$$\begin{array}{lll} \min & c_0 t_0 + c_1 t_1 + \ldots + c_n t_n & \\ s.t. & t_j - t_i \geq w_{ij} & (i,j) \in F \\ & (t_j - t_i \geq w_{ij}) \vee (t_k - t_h \geq w_{hk}) & ((i,j),(h,k)) \in A \\ & t_0 = 0 & \end{array}$$

This problem can be therefore formulated as a particular *disjunctive program*, i.e. a linear program with logical conditions involving operations “and" ($\wedge$, conjunction) and “or" ($\vee$, disjunction), as in [Balas 1979] and generalizes disjunctive graph [Roy 1964] and some of its extension [Mascis 2002]. A solution to this problem can be conveniently represented as a temporal network $G = (N, F \cup S)$ where N is a set of nodes of the graph representing events and F and S are two sets of weighted directed arcs. This is a general representation that, even though it does not allow representing some problems (such as Resource Constrained Project Scheduling), it can model most well known scheduling problems. In our approach the temporal network representation is used to represent solutions, and subsequently for generating instances.

A stochastic version of a scheduling problem can be represented by considering the length of the arc as a probability function $f(p_{ij})$. By $f_s(p_{ij})$ we denote the s-th sample of the distribution $f(p_{ij})$.

We use the following notation to represent the uncertainty and the dynamic evolution of a dynamic scheduling problem. Let $\delta = (i, j, f(p), t)$ be an *exogenous*

change in the problem where t is the time in which the change in the distribution is revealed to the scheduler, and i, j and $f(p)$ are the starting, ending node and the new probability function for the arc length, respectively. Let Δ_t be the set of exogenous changes revealed at time t' where $t' \leq t$. Hence, Δ_∞ represents all the exogenous changes that will happen during the execution of the dynamic instance while $\Delta_{-\infty}$ represents the information available in the planning phase.

Let $\bar{\Delta}_t$ be the set of changes caused by the solver until time t. These internal changes are algorithmic decisions carried out before time t that cannot be undone, i.e., scheduling decisions such as adding one alternative arc (i, j) to the current selection S.

We use the notation that if there exist two arcs connecting the same pair of nodes i and j then the most recent arc overwrites the old arc. Moreover, observe that, setting the weight to $-\infty$ is equivalent to removing the arc from the graph.

Hence, an instance for the dynamic scheduling problem I is the pair $I = (G, \Delta_\infty)$. An instance contains all the exogenous future changes that will take place in the system. However, when solving a single deterministic scheduling problem the algorithm faces only the information that has been revealed so far since the future information is not available to the solver. The *state of the system* Σ_t at time t is given by the tuple $\Sigma_t = (G, \Delta'_t, t)$ where the set $\Delta'_t = \Delta_t \cup \bar{\Delta}_t$ contains the exogenous changes Δ_t known at time t and the (possibly empty) set of all the algorithmic changes $\bar{\Delta}_t$ introduced by the algorithm in the previous iterations. Observe that, in general, Σ_t and Σ_{t+1} can lead to two different deterministic scheduling problems even if Δ'_t coincides with Δ'_{t+1}, because moving from time t to time $t+1$ may cause some operation to enter the frozen period and thus leading to two different deterministic instances. This can for example happen in a single machine case, where an operation is scheduled to start at time $t_{frozen} + 1$, but a better solution exist where the operation is scheduled later.

Let $S((G, \Delta'_t))$ be a solution to a deterministic scheduling problem with the information available at the state $\Sigma_t = (G, \Delta'_t, t)$.

Let $z^*((G, \Delta'_t, t), (G, \Delta''_t, t))$ be the value of the objective function obtained when applying the optimal deterministic solution computed for the system state (G, Δ'_t, t) at time t to a possibly different system state (G, Δ''_t, t).

Theorem 4.3.2. $z^*((G, \Delta'_t, t), (G, \Delta'_t, t)) \leq z^*((G, \Delta''_t, t), (G, \Delta'_t, t))$

Proof. Assume $z^*((G, \Delta'_t, t), (G, \Delta'_t, t)) > z^*((G, \Delta''_t, t), (G, \Delta'_t, t))$ then there exists a solution with a lower objective function value than $z^*((G, \Delta'_t, t), (G, \Delta'_t, t))$ which was defined to be optimal. Thus we must reject the assumption. □

Observe that the optimal solution of the $I = (G, \Delta_\infty)$ problem $S^*(G, \Delta_\infty)$ may not be obtained by optimally solving all the deterministic problems arising as soon as the uncertainty is revealed $z^*((G, \Delta'_t, t), (G, \Delta'_t, t)), \forall t$. To show this, it is enough to observe that, as in analogy with greedy algorithms, the solver may be forced to take some decision (i.e., adding either arc (i, j) or (h, k) to S) before some relevant

information about that decision is actually known. And, once the decision is taken, it cannot be undone.

$z^*((G, \Delta_\infty, \infty), (G, \Delta_\infty, \infty))$ is the optimal value of the ex-post optimization, i.e, when all the uncertainty is known in advance, and it is a lower bound for the best attainable solution (Theorem 4.3.2).

$z^*((G, \Delta_{-\infty}, 0), (G, \Delta_{-\infty}, 0))$ is the optimal value of the ex-ante optimization, i.e, when no uncertainty is known in advance. Observe that it does not necessarily give an upper bound for the optimal solution of I. To show this, it is enough to consider a set $\Delta_\infty = \{\delta_t, \delta_{t'}\}$, where the exogenous change δ_t overwrites the original value $f(p_{ij})$ and $\delta_{t'}$ brings it back to the original value. The ex-ante optimal solution therefore coincides with the ex-post optimal solution, while the optimal solution computed at time t may be worse.

4.4 Literature and Applications

Dynamic scheduling problems are an interesting practical extension of classical scheduling problems but they are not deeply investigated [Ouelhadj 2009]. Among the possible approaches, the simulation based are often applied to address the rescheduling problem [Ramasesh 1990] since they combine classical scheduling problems with widely applied techniques such as the discrete-events simulation.

The recent surveys by Vieira et al. [Vieira 2003] and Aytug et al. [Aytug 2005] review these approaches and applications.

The common simulation-based approach makes it possible to either simulate field disruption or gather them from the real life application. The architecture of simulation-based approaches is usually composed of a solver (which solves the deterministic scheduling problem) a controller (which decides whether or not the plan has to be updated) and a simulator (which simulates the world and decides the exogenous events). Some papers have proposed general purpose rescheduling frameworks that correspond approximately to the same architectural approach. Different online strategies are evaluated, but typically:

- The framework can be used to assess the *robustness* of a solution, i.e., to evaluate the effect of possible disruptions on the objective function.
- The *no rescheduling* strategy consists in applying the plan computed offline in face of all the real-time perturbations. This is the simpler strategy since it does not require any rescheduling during the schedule execution.
- The *repair* strategy, which usually allows only rescheduling of the changed job while keeping the rest of the schedule fixed.
- The *rescheduling with penalties* strategy allows changes to the planned schedule in response to real-time perturbations, however the schedule which is currently under execution impose some constraints on the newly produced plan.

These constraints can be in form of limits on the number of changes or penalties in the objective functions. In other words, with this strategy, when solving a new instance in real-time the objective function is somewhat different with respect to the offline objective function. Rescheduling with penalties strategy takes *stability* issues into account, that may arise in some applications if the plan is changed too frequently.

- The *full rescheduling* strategy allows changes to the planned schedule in response to real-time perturbations, without taking the previous schedule into consideration. The full rescheduling strategy tends to maximize the *efficiency/solution quality*.

- A less common approach is the *progressive scheduling* approach which consists in building the plan in real-time while it is executed, i.e., progressively solving with very short time windows.

In their paper, Honkomp et al. [Honkomp 1999] model two chemical processes as a State Task Network (STN). The offline scheduling problem is solved using CPLEX to solve a MILP formulation of the problem and with a Bayesian heuristic. Different online strategies are evaluated; the no rescheduling, rescheduling with penalties and full rescheduling. The evaluated objective functions is the deviation from the deterministic objective function.

Cowling and Johansson [Cowling 2002] considered the classical single machine problem $1||\Sigma C_i$ which the SPT rule solves to optimality. Perturbations happen only in the first half of the schedule, and when information arrives three strategies are evaluated: the no rescheduling, a repair strategy and a full reschedule according to shortest processing time (SPT) rule. As evaluated objective functions they consider a linear combination of utility (sum of completion times) and stability.

Pfeiffer et al. [Pfeiffer 2007] developed the same simulation-based framework and addressed three different problems: *i)* a single machine minimizing the average flow time with releases, *ii)* a small flexible job shop problem (5 machines and 8 jobs) and *iii)* industrial application with the objective of minimizing the tardiness. The offline solution is computed using heuristics, and the rescheduling process is triggered either by a fixed rescheduling interval (periodic rescheduling) or when a threshold between the planned and simulated solutions is exceeded. They evaluated the effects of evaluation of machine breakdowns and stochastic processing times. In their tests, the evaluated objective functions takes both stability and efficiency into consideration.

Bidot et al. [Bidot 2009] consider a dynamic job shop scheduling problem which is solved offline using the ILOG Scheduler. As online strategies they consider a full rescheduling (where the rescheduling process is activated by a trigger) and progressive techniques. The tests are carried out using (10x10) benchmark instances and only efficiency is evaluated in the tests.

Recently, Rasconi et al. [Rasconi 2010] consider a full rescheduling and a rescheduling with penalties strategies. The tests are carried out on RCPSP/max

benchmark instances and evaluate CPU requirements, frequency of rescheduling, rescheduling success rate as well as stability and efficiency.

The simulation based approach is also applied in this chapter. In the following section we introduce in details of all the components of the proposed framework.

4.5 Architecture

This section describes the implementation of the dynamic rescheduling framework proposed in this chapter. The framework is implemented in Java version 1.6, but has been shown to work in Java 7. It uses the statistical package R version 2.13.1 for generating plots, but functions without it if no plots are needed.

The real-time rescheduling problem is divided into three subproblems: (*i*) Determining and implementing a policy for rescheduling. (*ii*) Invoking a solver on the reoptimizeable parts of the solution. (*iii*) Adapting the solution based on the reoptimization. The proposed system addresses the resolution of the three subproblems. Figure 4.2 presents the framework for dynamic rescheduling, which is composed of a Simulator, a Solver, an Instance Writer and Solution Reader module. Subproblem (*i*) basically consists in defining the behaviour of the framework (when to invoke the reschedule process) while respecting constraints posed by the real life application. Subproblems (*ii*) and (*iii*) are tackled by the Instance Writer and Solution Reader modules of the framework.

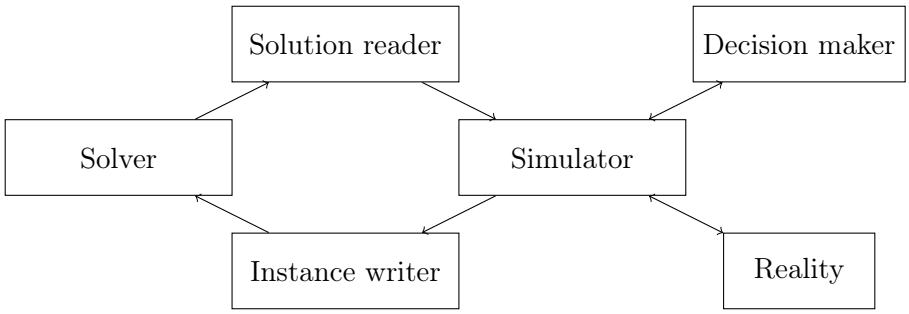

Figure 4.2: Architecture of the Dynamic Rescheduling Framework.

4.5.1 Reality Module

In laboratory experiments the Reality module is in charge of replicating the real world. Thus it contains the "real" information about uncertainty and probability functions and provides it to the Simulator in form of exogenous events Δ_t. Observe that all the other modules are not aware of how such exogenous events are generated. Typical changes to a distribution associated with an uncertain event makes the event become less uncertain as more information becomes available. This process terminates when all stochasticity is removed and the actual duration of each operation is finally known. Depending on the application the actual duration can

be known in advance with respect to the starting time of the operation or while the operation is being processed.

If the framework is used under real life conditions, the reality module receives the events via an interface, and transmits them to the simulator module.

4.5.2 Simulator Module

The Simulator implements the rescheduling policy, i.e., to decide when the Solver should be invoked. This is done by keeping a representation of the graph G from the system state Σ_t. The state can change as a result of an *event*. We distinguish three types of events:

End of the rescheduling process. This event is communicated to the Simulator by the Solution Reader module when a new scheduling solution is available.

Exogenous change. This event is communicated by the Reality module signifying a changed distribution of an operations duration.

Time progress. This event is internally generated by the Simulator and consists in updating the system state Σ_t to $\Sigma_{t'}$, i.e., from time t to time t'.

To bootstrap the loop in Figure 4.2, the Simulator is fed an initial solution to the instance $(G, \Delta_{-\infty})$ computed in the planning phase. From this solution, the initial state $\Sigma_{-\infty} = (G, \Delta'_{-\infty}, -\infty)$ is created.

To analyse the stochasticity of the problem, the Simulator generates a predetermined amount x of Monte Carlo samplings by instantiating the probability functions associated with the uncertain events. These will be realised as a set of x different scenario graphs $G_s \in G_1 \dots G_x$. G_0 refers to the graph that would be converted to an instance if rescheduling is done. The Simulator then solves a series of longest path problems rooted in the source node to all other nodes in the graphs to obtain start times $t_{i,s}$ for each node in G_s. These values are then used to decide if a rescheduling is warranted, and to analyse the distribution of objectives.

Longest paths are calculated using a longest path version of the Bellman-Ford algorithm. It is able to detect possible positive length cycles that indicate infeasibility (i.e., an operation preceding itself). When an event Δ_t causes a decreased length of an edge (i, j) in a scenario graph G_s a simple check can verify if o_j has a changed start time $t_{j,s}$. If and only if that is the case, the longest path calculations are redone from scratch. If the change causes an increase, o_j is marked as changed in the Bellman-Ford algorithm, and the longest path algorithm is restarted.

The primary memory usage of the simulator is representing the scenarios $G_1, \dots, G_x$. To minimize this, G_0 contains the structure of the graph and G_s, $s \geq 0$ is represented as an array of start times $t_{i,s}$ and an array of process times $p_{ij,s}$

4.5.2.1 Triggers

A trigger is a function $T(\Sigma_t, t, t_{last}, \Delta_t \setminus \Delta_{t_{last}})$ that maps the current state of the system Σ_t, the time t, the last time a solver was run t_{last}, and the events $\Delta_t \setminus \Delta_{t_{last}}$

into a boolean value. When the condition of a Trigger is satisfied then the Simulator calls the Solver for a reoptimization.

Triggers can be grouped by type according to the data that trigger them. *Deterministic triggers* are triggered by changes in the deterministic data potentially passed on the solver G_0, while *stochastic triggers* work on the Monte Carlo trials on the graph representing the system state $G_1 \dots G_x$.

Though the dynamic rescheduling framework is not limited to these, the following types describe some of the most common deterministic triggers:

Changed makespan is triggering when the longest path from source to the sink node changes.

Lateness of an operation is triggered when the longest path from source node to the corresponding node plus the duration of the operation exceeds the due date of the operation.

Infeasibility of the solution originally provided by the solver.

Sum of lateness exceeding a given value.

Stochastic triggers are based on a boolean expression that can be evaluated on each of the scenario graphs $G_1 \dots G_x$, and these can usually be considered deterministic triggers themselves. They trigger when the boolean condition C is satisfied with a given percent chance with confidence p. Examples of these:

- Trigger when you can say with 95% confidence that there is at least 10% chance of infeasibility.
- Trigger when you can say with 95% confidence that the objective will be worse than the objective the solver achieved during last rescheduling.
- Trigger when any chance of infeasibility is detected. This detection can be made more accurate for some applications by forcing G_1 to contain the worst case durations for all operations.

Custom triggers can be implemented by adhering to a provided interface.

4.5.2.2 Human Interaction

In some applications the Simulator could work with no human intervention or supervision, while in other applications a decision maker could decide whether or not a new rescheduling should be started and supervise the solutions found by the Solver. More often, the presence of a decision maker should be considered. Therefore there is the need to find a suitable way to represent the current solutions of the system and give the decision maker the overwrite possibility (i.e., the chance to change the decisions taken by the automated system).

A possible way to provide information on the solution to the supervisor is to make use of stochastic Gantt charts, as the ones shown in figure 4.3, 4.4 and 4.5.

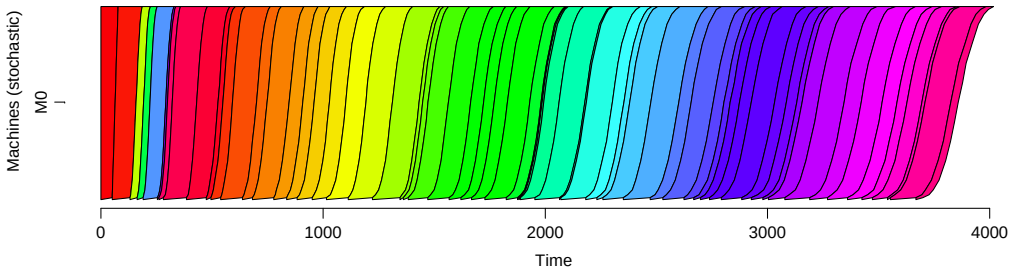

Figure 4.3: A stochastic Gantt chart for a SMWCTP instance with 60 jobs.

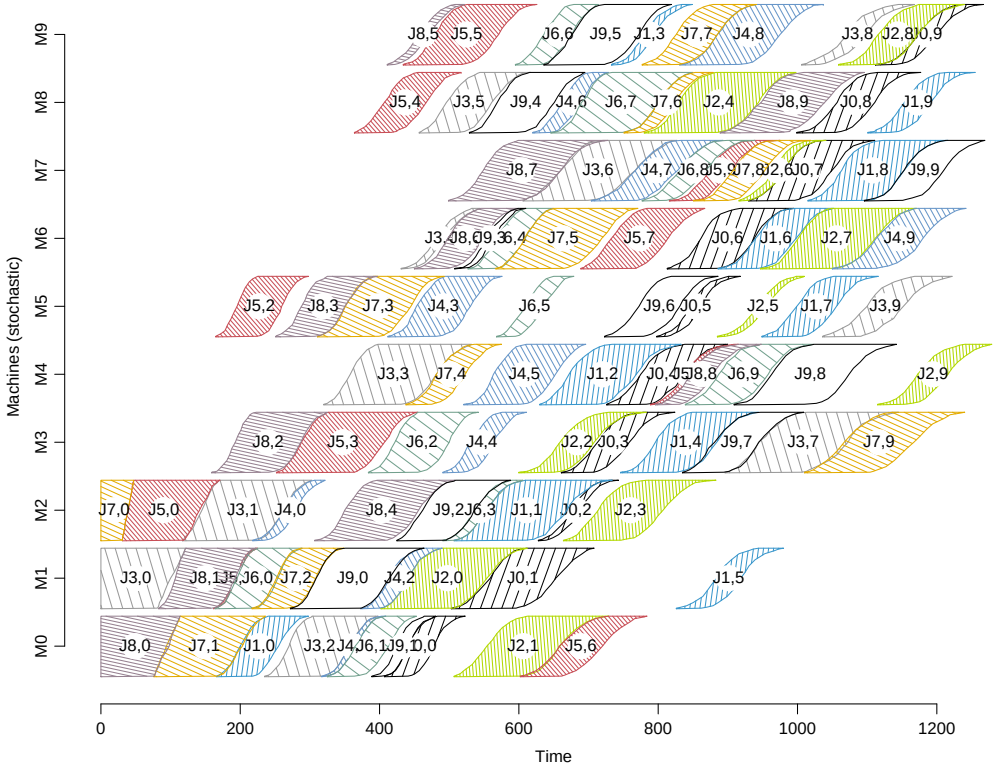

Figure 4.4: A stochastic Gantt chart for the FT10 job shop instance.

These plots are based on predictive Gantt charts introduced in [Baker 2009].[1] The stochastic Gantt charts are useful for displaying stochastic schedules. They are drawn as regular Gantt charts, but the boxes signifying an event occurring with certainty at a given time, have been replaced by a shape representing the approximate likelihood of an operation being processed at a given time.

The height of the row representing each machine is not explicitly drawn, but can be seen as the topmost and bottommost part of shapes representing the operations

[1] We would like to thank Geoff Robinson and Andreas Ernst for providing R code we modified to display the shown Gantt charts. No citable articles have been published yet.

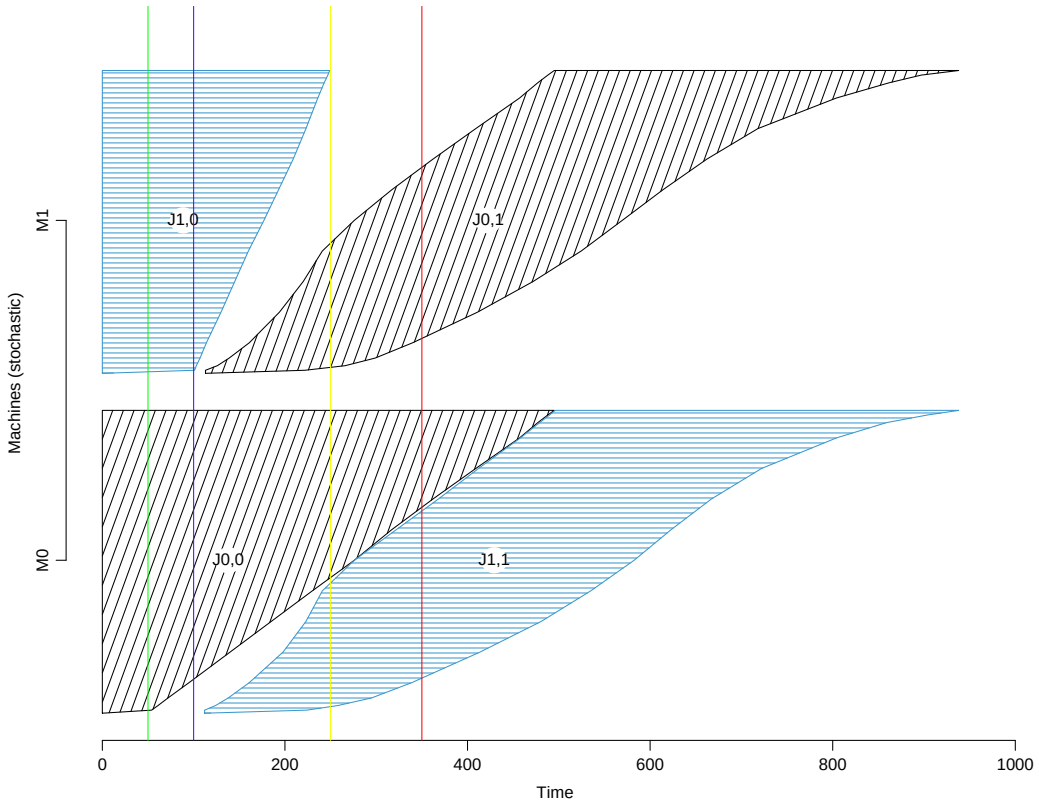

Figure 4.5: A small stochastic Gantt chart for a two machine job shop problem.

belonging to that machine. Figure 4.5 is a solution to a problem with two machines and two jobs ($J1$,$J2$) each with two operations. One color is assigned to each job. The red vertical line at time 350 intersects the shapes representing three operations labelled $J0,1$, $J0,0$ and $J1,1$ respectively. The length of the red line in the row corresponding to machine $M1$ can be divided into three parts: The area above the shape representing $J0,1$ in the $M1$ row, represents the chance that $J0,1$ has not started at time 350, the area intersecting represents the chance that the operation $J0,1$ is currently running, and the are below represents the chance that $J0,1$ has finished at time 350. From the part of the red line at time 350 intersecting the row representing machine $M0$, we can deduce that there is a chance $J0,0$ is still running, a bigger chance that $J1,1$ is running, and finally a chance that no operation is occupying machine $M0$ at that time.

The other lines in Figure 4.5 are draw at the following points: The green line at time 50 marks the point where $J0,0$ can potentially finish. The blue line at time 100 marks the time when $J1,0$ can potentially end, and thus a lower bound on when $J1,1$ can start. Finally the yellow line at time 250 marks the time when $J1,0$ must have ended. The gap between the shapes on machine $M0$ represents a possibly empty machine due to $J1,1$ being constrained by the previous operation in the same job.

Each shape is drawn based on linear interpolation between points drawn for the

following inputs to the quantile function: .01, .025, .05, .1, .2, .3, .4, .5, .6, .7, .8, .9, .95, .975 and .99. Note that the first and the last percentile are not represented to avoid visual artefacts on long tailed and unbounded distributions. This is the reason for the blue line in Figure 4.5 not touching the shape representing $J1, 1$.

If other visualizations are required, the decision maker can request density plots of timings of any operation, or for an objective function. This can be done for multiple solutions, enabling comparisons such as the one depicted in Figure 4.6.

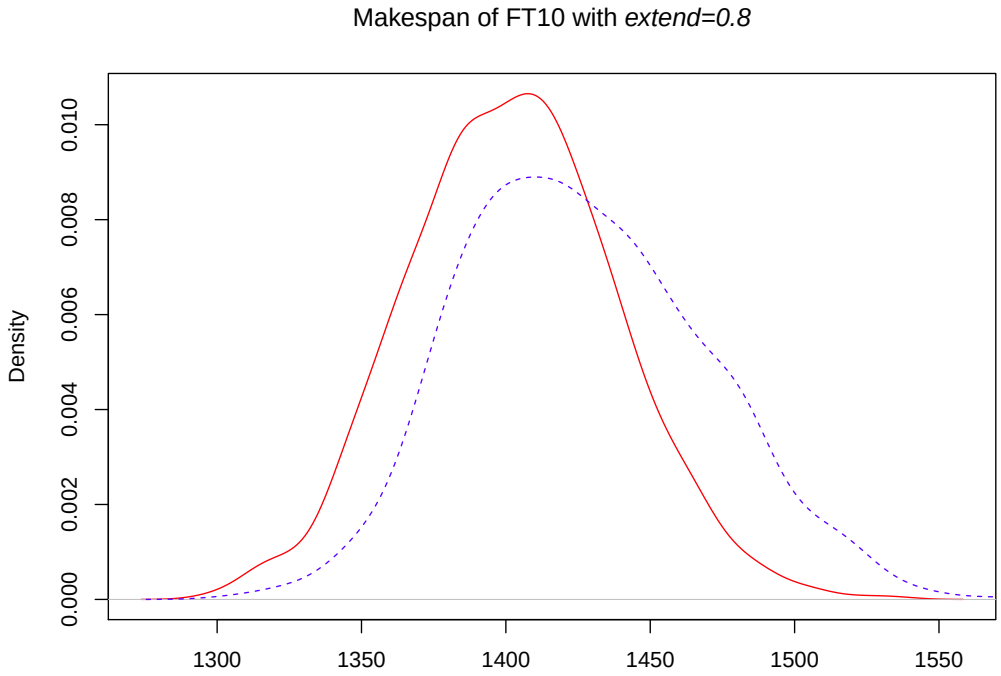

Figure 4.6: Density plot of the makespan for two solutions to the FT10 job shop instance with added stochasticity ($extend = 0.8$). The red solid line represents a superior solution to the blue dotted line given that we want to minimize the makespan. This can be seen as a shift toward lower makespans (left on the x axis).

For guided example of information available to the decision maker during the solution of a job shop problem, see Appendix B.

4.5.3 Solution Reader and Instance Writer Modules

The Solution Reader and Instance Writer modules must be implemented separately for each Solver that is to be used with the framework.

The Solution Reader module must generate a temporal network $G = (N, F \cup S)$ from the instance file and the solution produced by the deterministic solver. It must associate any provided distributions to their corresponding edges.

The Instance Writer module is responsible for creating a deterministic instance based on the simulation of the current system state Σ_t. The primary challenge when implementing the Instance Writer module is the progressing time during simulation. The Instance Writer is provided a list of events that are considered locked in time due to them having started or being in the frozen horizon. These events can be transferred to a graph $G_{temp} = (\emptyset, \emptyset)$ that will be provided to the read module after the Solver has been executed. Locked events should never be moved by the Solver, and this should be enforced by the writer module. Common ways of doing this includes leaving them out of the instance and mimicking their presence by enforcing release times on other operations, or alternatively introduce deadlines and release times that fixes the original operations in time. The choice is determined by the ability of the solver to handle the constraints needed.

Example I/O files for the three problems analysed by the framework in this thesis can be found in appendix A. Section A.2 is of special interest for the following example as the files correspond to a larger instance of the same problem.

4.5.3.1 Reader and Writer Modules for a Job Shop Problem

To illustrate the function of the reader and writer module, a small job shop example is provided. A job shop problem $(J2|r_j|C_{max})$ is solved in the planning phase, providing start times for each operation, which are translated into an ordering of operations on each machine.

Figure 4.7 shows the graph G the solution reader is building. Solid/fixed edges F can be drawn based on the instance file, as they are always a path from *source* to *sink* visiting each operation $o_{i1} \dots o_{im(i)}$ in a job J_i in that order. The dotted/selected edges S are drawn as a path from the first to the last operation on each machine based on their start times from the solution. The length of the edges from *source* corresponds to release times,[2] while the rest of the edges gets their length p_{ij} from the previous operation's processing time. To make the graph stochastic, each edge is also associated with the probability distribution of $f(p_{ij})$ of their length, read from a separate input file.

When a rescheduling is required, the graph from Figure 4.7 is split along the dashed line representing the edge of the frozen period. Every node to the left $(J1, 0)$ are considered frozen, and copied to G_{temp} represented in Figure 4.8. Next a dummy node is constructed for each machine ($M1_d$ and $M0_d$). The purpose of the dummy nodes is to represent the new first operation on each machine after the frozen period when the rescheduling is done. Then all edges terminating in a frozen node are copied to G_{temp}, and all edges from F crossing the dashed line are copied to G_{temp} with their destination replaced by the appropriate dummy node. Finally the dashed edges representing new[3] release times are added to each dummy node, ensuring that they never are scheduled earlier than the end of the frozen period.

[2] The length is 0 if no release time exist.

[3] If release times are already present, the maximum release time is enforced.

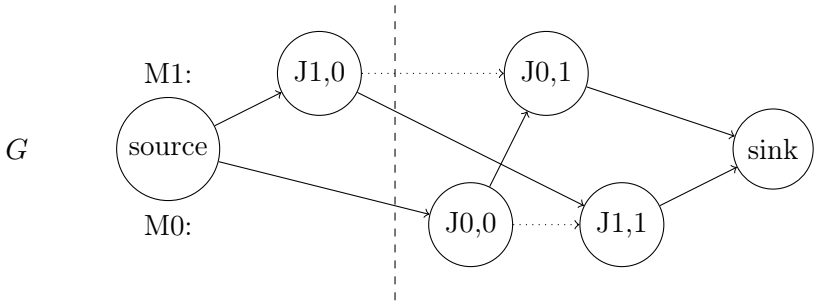

Figure 4.7: The graph G for a small job shop problem. Each row corresponds to a machine, and the horizontal position is indicative of start times. The solid arrows corresponds to the fixed edges F, the dotted arrows correspond to the selected edges S. The operations left of the dashed vertical line are considered frozen and may not be moved.

The instance of the rescheduling problem is then created as the original instance with the following modifications:

- Processing times are modified based on a sampling strategy mapping probability functions $f(p_{ij})$ to a deterministic value p'_{ij}.
- Release times are introduced for each non-frozen operation, ensuring they are scheduled after the frozen period.
- Processing times for each frozen operation is set to 0, making them easy to schedule without having any impact on the rest of the solution. If the solver does not minimize the start times of these effectively, care must be taken when reading the solution.

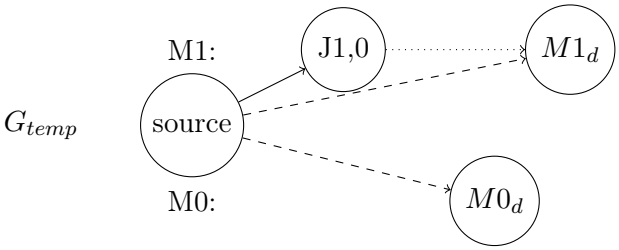

Figure 4.8: The graph G_{temp} for a small job shop problem.

When the solver has terminated, the solution is read using the solution reader module, ignoring operations with a processing time of 0. Then all edges from G_{temp} is copied into the solution, replacing edges already present and replacing $M0_d$ and $M1_d$ with the new first operation on their respective machines.

4.5.4 Solver Module

The Solver is the optimization core of any rescheduling system. It should be able to solve the current rescheduling problem. The proposed dynamic rescheduling framework is solver independent once the proper communication can be established with the Solver, provided a method can be devised to ignore or lock parts of the instance that have already been locked as described in Section 4.5.3. Both exact and heuristic solvers can be interfaced with the framework. In Section 4.6 we show how a dispatching rule yielding the optimal solution or a truncated commercial solver can both be used as solvers within the framework, and in Chapter 5 we use a solver based on the alternative graph model.

4.5.5 Parameters and Input

The framework requires the following input:

Solver to use for rescheduling. This will also determine the read and write modules to use. If the solver is marked as deterministic, it will not be invoked twice on the same instance.

Deterministic instance in a form the desired solver can parse.

Stochastic instance in the form of probability functions for the non deterministic process times.

Sample instance is a mapping from probability functions $f(p_{ij})$ to an edge length p_{ij} that can be used to set the actual durations of process times if the framework is simulating without a link to a real life application. Sample instances thus represent the actual realization of a given stochastic instance (a scenario).

The framework must also be given the following application specific parameters:

Knowledge parameter $k \in (-\infty; 1]$ is used when the framework is not connected to a real life application. It describes when events are generated that reveal the actual durations of an operation's processing time to the framework.

A negative integer value denotes the amount of time units before the operation is scheduled to begin in the current schedule. A value in the interval $[0; 1]$ denotes the completion an operation must be at before the process time becomes known to the framework. $k = 0$ thus corresponds to generating the event when an operation starts, $k = 1$ when it ends and $k = -1$ the time unit before the operation is scheduled to start.

In case of negative k, gained information is retained even if the operation is rescheduled thereafter

Frozen period the amount of time ahead of the starting time of an operation, where it should be considered locked in the schedule.

Allowed CPU time the maximum amount of time the solver is allowed to use for reoptimization.

Frequency parameter $f \in [1; \infty]$ is a lower bound in time units between runs of the solver. A value lower than one will be treated as a 1 to avoid infinite cycling.

Finally the framework must be given information detailing how and when to reschedule. These inputs and parameters can be chosen with few hard constraints due to the problem being treated.

Number of samples $n \in [0; \infty)$ determines the number of graphs $G_1 \dots G_n$ created and maintained by Monte Carlo sampling.

Sampling strategy $g \in [0; 1]$ is used to generate a deterministic instance for the solver based on the stochastic instance under consideration. Each stochastic process time must be made deterministic. For bounded distributions this will be treated as reverse lookup in the cumulative distribution function. Thus $g = 0$ corresponds to the shortest possible duration, and $g = 1$ corresponds to the longest possible duration and $g = 0.5$ is the median of the distribution. For unbounded distributions, a maximum value must be enforced to avoid generating infinite values.

Triggers must be implemented beforehand, and can be enabled by a parameter.

Rolling Horizons can be handled if implemented in the instance reader and writer, and the solver must be able to solve the partial instance.

4.5.6 Requirements for Application of the Framework

In order for a problem to be treatable by the framework, it must have solutions representable as a temporal network implemented by a graph G. The feasibility of a solution must be derivable from the temporal network. Note that some resource constraints could potentially violate this constraint and thus cannot be tackled by the framework currently, if the solution might become infeasible due to resource constraint violations.

Clearly the solver must be able to solve the problem under consideration. If rescheduling is to be done, the solver must support *either* fixing the frozen parts of the schedule *or* removing/ignoring already executed or frozen parts of the schedule. The latter can be achieved by setting durations of frozen operations to zero and enforcing a release time on the following operations.

The probability functions provided must support a sample function yielding values for the Monte Carlo trials. Furthermore, a selection scheme must be implemented, that maps a value $g \in [0; 1]$ to a valid sample of the distribution.

4.6 Computational Experiments

The computational experiments reported in this chapter are based on two different scheduling applications. Namely we consider the Single Machine Weighted Completion Time problem and the academic Job-Shop Scheduling problem.

To illustrate the functionality of the framework, we perform an exploratory analysis of each of the two problems, followed by the analysis of an artificial test case. In Chapter 5 we use parts of the framework on a real life instance of train scheduling.

All the tests have been carried out on a Intel(R) Core(TM) i5 CPU 650 @ 3.20GHz.

4.6.1 Single Machine Weighted Completion Time (SMWCTP)

The single machine problem we consider is the classical $1||\sum w_i C_i$ single machine problem with the objective of minimizing the sum of the weighted completion time of each job. Given a job $J_i = \{o_{i1}\}$, let p_i and w_i denote its processing time and its weight, respectively. Given a solution, we denote the completion time of job J_i C_i. The problem is known to be polynomially solvable [Smith 1956] by applying the Weighted Shortest Processing Time (WSPT) rule, that is to sequence jobs according to their non-decreasing ratios p_i/w_i.

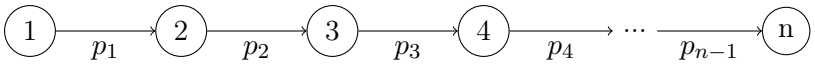

Figure 4.9: The graph representation of the solution to the single machine weighted completion times problem

Figure 4.9 shows how a solution of the SMWCTP can be represented using $G = (N, F \cup S)$ and thus communicated to and from the Solver.

We randomly generate 100 stochastic instances for computational experiments as follows. We consider 10 randomly generated instances of SMWCTP, each having a number of jobs randomly selected as $n \in [10; 100]$. Each operation o_i has a weight $w_i \in [1; 100]$ and a processing time $p_i \in [1; 100]$. All values are uniformly likely in each interval. We then create stochastic instances by changing all of the process times into a uniform distribution ranging from its original duration $p_{i,j}$ to $p_{i,j} + p_{i,j} \cdot extend$ for a given value of the *extend* parameter. In our test, we set $extend \in \{0.1, 0.2, \ldots 0.9, 1.0\}$. For each of the 100 stochastic instances, we create 4 *sample* scenarios by sampling each of the distributions associated with the process times. Sample instances are given to the simulator during simulations, and are used for generating actual values for each distribution such that consistent process times are used across multiple runs of the framework. Therefore each stochastic instance is evaluated four times per run of the framework, one for each scenario. As each stochastic instance is created with a different *extend* parameter, their objective values are expected to differ significantly. To make the data comparable, we calculate

the optimal value of the ex-post optimization $z^*((G, \Delta, \infty), (G, \Delta, \infty))$ and calculate the deviation of each solution from this. Overall, 10 instances of SMWCTP have been extended into 100 stochastic instances, which are subsequently evaluated in 4 different instantiations (using the 4 scenarios).

During the experiments we tested 15 different "test case" configurations (i.e., parameters that are influenced by the real life application) by setting the knowledge parameter $k \in \{-400, -200, -100, -10, 0\}$ and the frequency parameter $f \in \{1, 10, 100\}$. The sampling strategies were set to $g \in \{0, 0.25, 0.5, 0.75, 1\}$ bringing the number of test cases up to 75. Observe that there is no need to test cases with $k > 0$ since any value of $k \in [0; 1]$ would yield the same deterministic instances: $k \in [0; 1]$ indicates that information about an operations process time is revealed during the operations execution. This means that the operation is already frozen at that point in time and cannot be moved, and for SMWCT the residual problem is just sorting the remaining operations using the WSPT rule. $k = 0$ thus corresponds to the optimal ex-ante solution. In the following results $k = 0$ will not be considered in any averaged values.

Furthermore we used 3 Triggers:

$T_{objective}$ Trigger when the objective function changes.

$T_{change10}$ Trigger when the sum of absolute values of changes in the process times exceeds 10.

$T_{change100}$ Trigger when the sum of absolute values of changes in the process times exceeds 100.

For each of the triggers, the f parameter provides a lower bound on the time between rescheduling. Thus if $f = 10$ and $T_{objective}$ detects a changed objective at time $t = 1$ and $t = 2$, 10 time units will still pass between reschedules.

In total we have 225 configurations of the framework, 100 instances and 4 scenarios. This creates 90000 different values of the objective function.

4.6.1.1 Results (SMWCTP)

Figure 4.10 shows the average quality of the solutions obtained, as a function of the sampling strategy g for set values of k and f respectively. Figure 4.10 (a) shows that guessing a low process time p_i seems favourable for low values of f, i.e., when rescheduling frequently, while less frequent optimizations favour g values closer to 0.5 and thus guesses closer to the expected value of p_i. The time at which the framework becomes aware of actual durations k (Figure 4.10 (b)) shows a tendency to favour lower values of g. For comparison, the optimal ex-ante solution are approximately 5.2% worse than the optimal ex-post solution as can be seen in Table 4.3 later.

Table 4.1 shows that rescheduling at every change of a duration ($T_{objective}$) when the f parameter allows leads to solutions that deviate from the optimal ex-post solution by 0.41%, when averaged over all values of k, *extend* and g. If an exogenous cost is associated with each rescheduling, rescheduling only if the change in duration

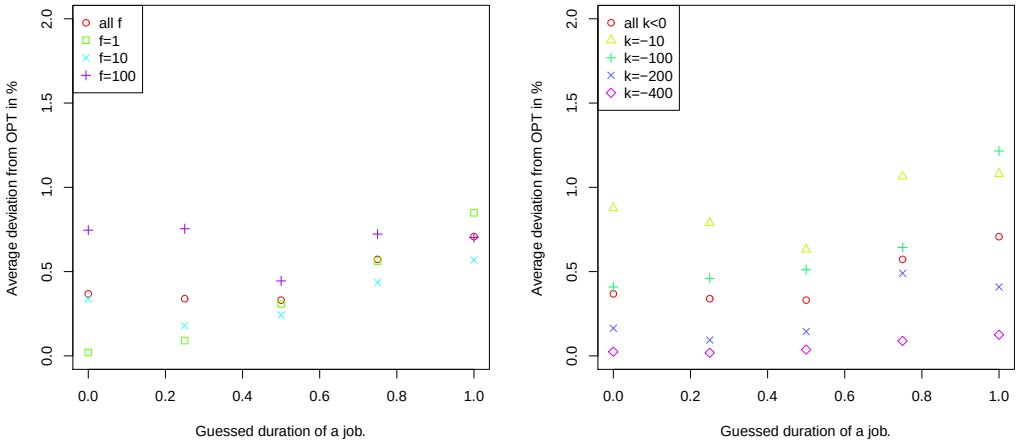

Figure 4.10: Solution quality as a function of the guess parameter g for the single machine weighted completion time with the $T_{objective}$ trigger. The left plot averaged over all k and the right plot averaged over all f.

	$T_{objective}$	$T_{change10}$	$T_{change100}$
Reschedules	31.0	19.7	4.8
Average deviation	0.41%	2.26%	38.89%

Table 4.1: The average number of reschedules and the average deviation from the optimal solution when applying the three triggers T_1, $T_{change10}$ and $T_{change100}$.

exceeds 10 time units ($T_{change10}$), becomes interesting as it saves 11.3 reschedules on average and obtaining solutions with 2.26% deviation from the ex-post optimum on average. The exogenous cost must be significant to justify using $T_{change100}$, as the average deviation increases to 38.89%.

f	1	10	100
Average deviation	0.26%	0.38%	0.59%

Table 4.2: Average deviation in weighted completion times for different values of rescheduling frequency f with the trigger $T_{objective}$.

Table 4.2 shows that the difference between rescheduling every one and ten time units is small compared to other sources of increase in solution quality. Even a rescheduling every 100 time units provide comparable quality.

k	-400	-200	-100	-10	0
Average deviation	0.06%	0.22%	0.54%	0.81%	5.16%

Table 4.3: Average deviation in weighted completion times for values of k.

Table 4.3 shows an increase in deviation from the ex-post optimal solution quality as k approaches 0. The major spike in deviation for $k = 0$ is due to the framework being forced to used the ex-ante optimum.

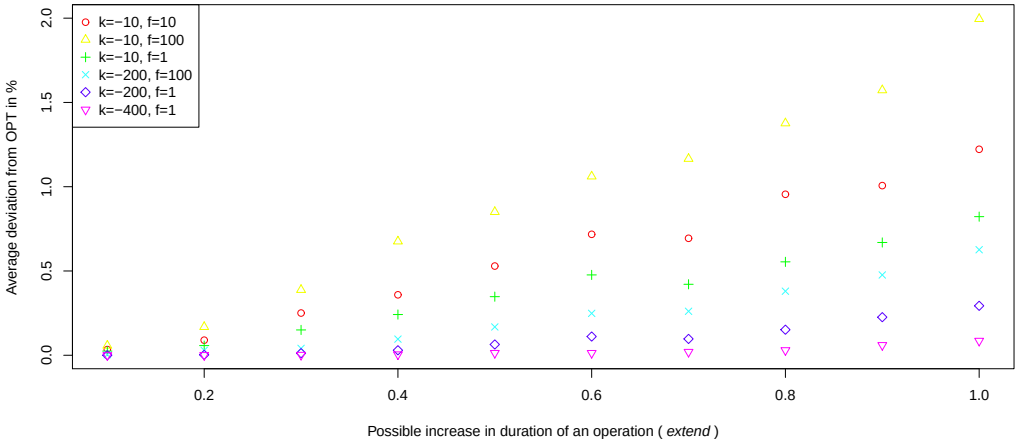

Figure 4.11: Solution quality as a function of the *extend* parameter for set values of k and f with the trigger $T_{objective}$.

Figure 4.11 shows effect of increased stochasticity on the solution quality. It is clear that solution quality rises significantly as k falls. Enforcing a pause of 100 time

units between each rescheduling ($f = 100$) comes at a small cost when $k = -200$, but the cost of increasing f rises when $k = 10$, presumably because the framework sometimes gets events but is unable to react on the due to a recent rescheduling when $f = 100$.

4.6.1.2 Test Case Analysis

Observe that the stochasticity *extend*, the knowledge k and the frequency f parameters are usually determined by the application. For this test case we will assume that $extend = 0.8$, $k = -10$ and $f = 10$, respectively.

The choices left to the managers are which trigger and sampling strategy to adopt. Assume that possible triggers choices are $T_{objective}$ and $T_{change10}$ and $g \in \{0, 0.25, 0.5, 0.75, 1\}$.

	$g = 0.0$	$g = 0.25$	$g = 0.5$	$g = 0.75$	$g = 1.0$
$T_{objective}$	2.00%	0.95%	0.96%	1.35%	1.72%
$T_{change10}$	2.37%	2.11%	2.94%	5.12%	3.36%

Table 4.4: Average deviation from optimal ex-post solution for $extend = 0.8$, $k = -10$ and $f = 10$.

By running the proposed dynamic rescheduling framework it turns out that the best configuration is $T_{objective}$ as expected and $g = 0.25$ since it provides the smaller deviation from the ex post optimum (as shown by Table 4.4).

If there is an exogenous cost associated with rescheduling, the problem becomes multi objective. $T_{objective}$ causes 41.6 reschedules on average while $T_{change10}$ causes 28.1. In this case two different solutions (each trigger with $g = 0.25$) become Pareto optimal, and the decision on which to keep must be left to a decision maker, or a scheme must be provided for comparing the two objectives directly.

4.6.2 Job Shop Scheduling Problem (JSP)

The classical job shop scheduling problem ($J||C_{max}$) is one of the most studied NP-hard problems [Pinedo 1995].

The Solver implemented is a standard constraint programming model for the classical job shop scheduling problem and it has been implemented using IBM ILOG CP Optimizer version 12.2. As solving to optimality takes prohibitively long, the allowed CPU time is set to 60 seconds and the solver uses a single worker thread.

Figure 4.12 shows the temporal network for a solution to a job shop problem with two jobs each having three operations being scheduled at three consecutive machines. The solid arcs represent precedence relations present in all solutions F, while the dashed arcs represent choices made by the solver S. Later rescheduling might change such sequencing decisions. The makespan C_{max} is calculated by finding the longest path from the source node to the sink node.

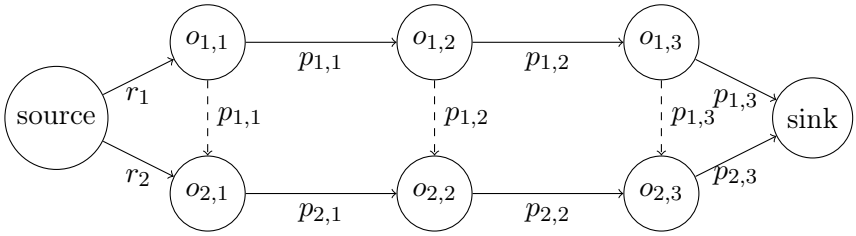

Figure 4.12: The graph representation of the solution to a simple job shop problem, obtained by ordering operations by their index i on each machine.

In order to generate our tests we consider the well known 10-job 10-machine benchmark instance FT10 [Muth 1963] as deterministic instance. We then create stochastic instances by changing all of the process times into a uniform distribution ranging from its original duration $p_{i,j}$ to $p_{i,j} + p_{i,j} \cdot extend$ ($extend \in \{0.1, 0.2, \ldots 0.9, 1.0\}$). For each of the stochastic instances, we create 4 scenarios by sampling each of the distributions associated with the process times. Thus we generated 40 scenarios in total.

As test cases configuration we use $k \in \{-400, -200, -100, -10, 0, 0.5, 1\}$ and $f \in \{1, 10, 100\}$, leading to 21 different application settings.

20 configurations of the framework, obtained by setting $g \in \{0, 0.25, 0.5, 0.75, 1\}$ and 4 Triggers, have been tested. The 4 Triggers are:

$T_{objective}$ Trigger when the objective function changes. For the job shop problem this corresponds to triggering when the makespan of the solution has changed.

$T_{frequency}$ Trigger as soon as the f parameter allows. And the solution has changed since last rescheduling.[4]

$T_{change10}$ Trigger when the sum of absolute values of changes in the process times exceeds 10.

$T_{change100}$ Trigger when the sum of absolute values of changes in the process times exceeds 100.

Again note that f provides a lower bound on the time between reschedulings regardless of the trigger used.

This setup corresponds to 16800 values of the objective function. With rescheduling this leads to 421608 runs of the solver, which have been achieved by parallelization over multiple machines.[5]

[4] Note that such a change might not affect the objective.

[5] Note that the final runs of the solver during each simulation is unlikely to consume the allotted time due to a reduced problem.

4.6.2.1 Results (JSP)

In Figure 4.13 we show the behaviour of the ex ante solution and 4 test cases configuration ($(k = 0, f = 10)$, $(k = -200, f = 1)$, $(k = -400, f = 1)$ and $(k = 1.0, f = 100)$) as the stochasticity influence increases (*extend*).

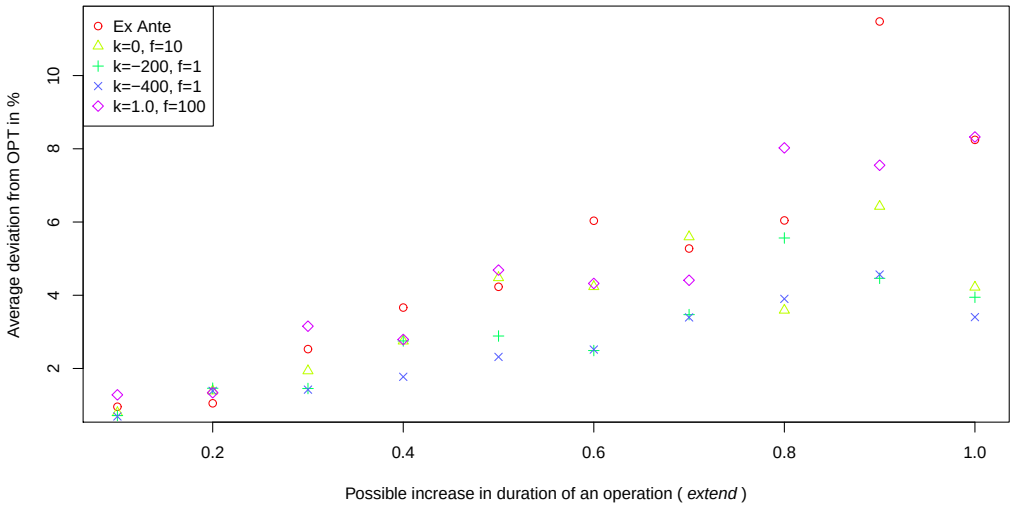

Figure 4.13: Gap between the ex-ante and ex-post solution compared with selected results by the framework.

Figure 4.13 indicates that there are situations where there exist a significant gap between the ex-ante solution and the ex-post solution. However for values of *extend* below 0.2 there seems to be no benefit to rescheduling on average, as the optimal solution found in the planning phase is better that what the truncated CP solver delivers. For higher values of the extend parameter there is a gap that can actually be exploited by allowing dynamic rescheduling. Figure 4.13 also indicates that an earlier knowledge of events (i.e., lower values of k) is beneficial. Besides, even if the actual duration is only known when the operation starts ($k = 0$) and the rescheduling is not too frequent ($f = 10$) still allows rescheduling to outperform the ex ante solution. The worst configuration tested for k and f ($k = 1.0, f = 100$) seems to have no consistent improvement over the ex-ante solution.

To illustrate the impact of the extend parameter we show in Figure 4.14 and 4.15 the difference in ex-ante and ex-post solutions for $extend = 0.2$ and 1.0 respectively. Observe that, for $extend = 0.2$ the two solutions coincides and the problem is thus stable enough to make rescheduling actions unnecessary. However, for higher values of the *extend* parameter the two optimal solutions differs, thus making rescheduling beneficial. For example of changes, see the first operation on machine M_8 in figure 4.15.

A clear benefit of obtaining knowledge earlier is expected. But when evaluating

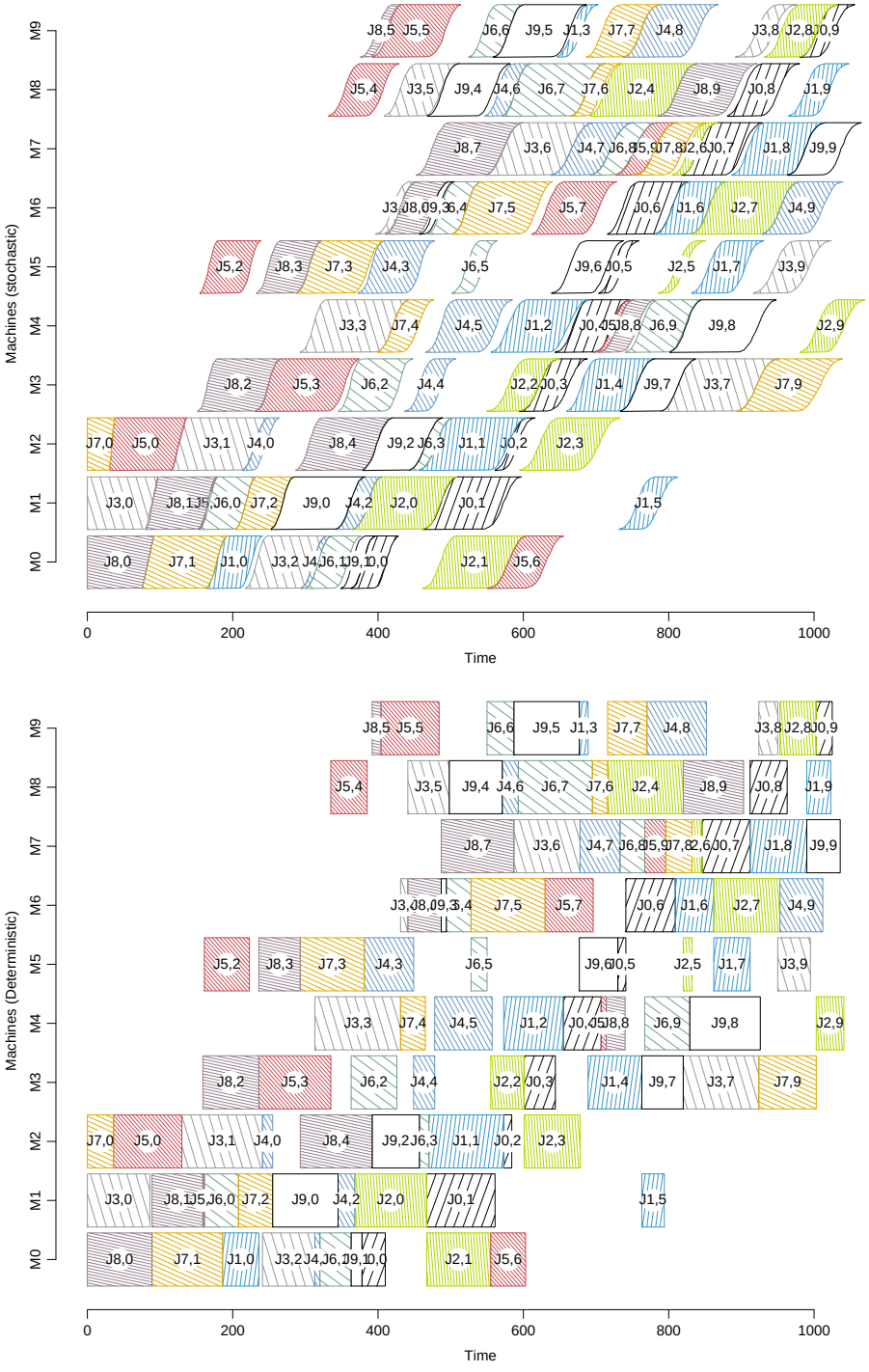

Figure 4.14: A stochastic Gantt chart for an ex-ante solution to a job shop instance for $g = 0.5$ and $extend = 0.2$ (top), and a deterministic Gantt chart for the ex-post solution to the same problem. **Note the conservation of the ordering of operations on each machine.**

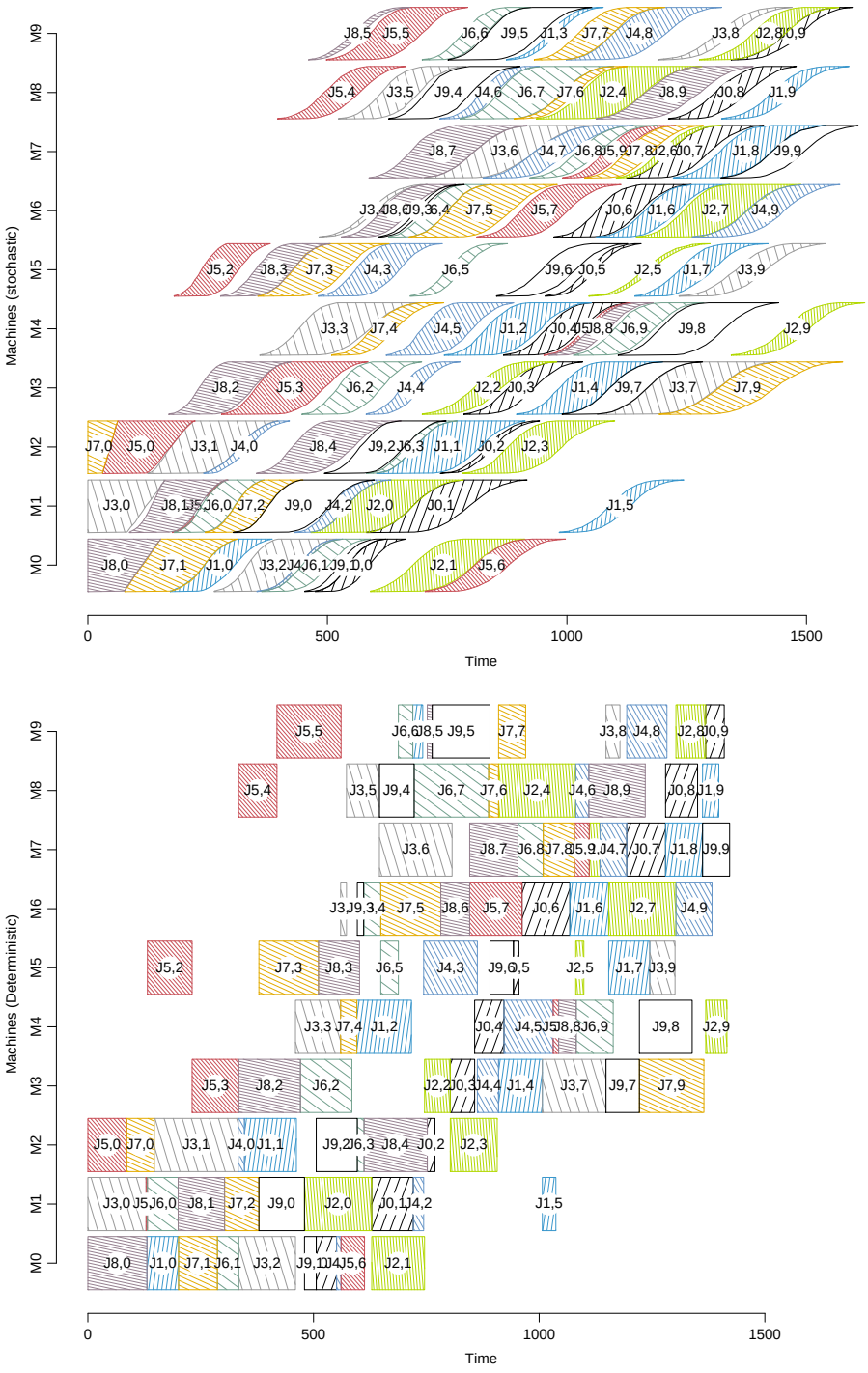

Figure 4.15: A stochastic Gantt chart for an ex-ante solution to a job shop instance for $g = 0.5$ and $extend = 1.0$ (top), and a deterministic Gantt chart for the ex-post solution to the same problem. **Note that the order of operations are not conserved on each machine.**

whether such information is worth the cost of obtaining it, the magnitudes of these benefits needs to be established. Figure 4.16 shows an approximately linear increase in solution cost as the k parameter increases. Contrary to the single machine problem, we observe little extra penalty for jobs being locked when the information about their durations are revealed ($k \geq 0$). This is due to other potentially affected operations possibly being unfrozen at that point.

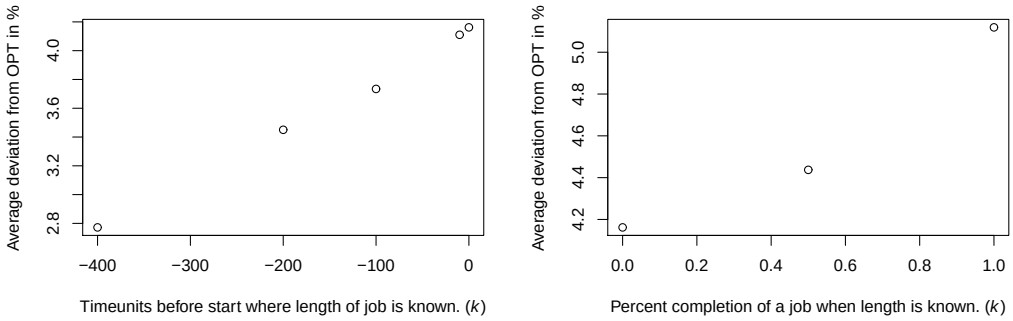

Figure 4.16: Solution quality as a function of the knowledge parameter k.

f	1	10	100
Average deviation	3.75%	3.83%	4.33%

Table 4.5: Solution quality as a function of rescheduling frequency f for the job shop problem.

Table 4.5 indicates the magnitude of the benefit of allowing frequent reschedules, but the data is averaged over all values for *extend*, g, s and k. On average, lower values for the frequency lead to better solutions.

To evaluate g's influence on the solution quality, we plot it for different values of k and f respectively in Figure 4.17. Contrary to the single machine problem we observe $g = 0.5$ results in the best results. This corresponds to passing expected values of probability functions to the solver.

If the job shop problem has exogenous cost associated with rescheduling or if time is limited, fewer reschedules can be obtained through the use of triggers. Table 4.6 shows that postponing rescheduling until process times have changed by at least 10 units ($T_{change10}$), on average produces solutions of the same quality as rescheduling when a change is encountered and the f parameter allows ($T_{frequency}$) while using significantly fewer reschedules. Rescheduling only when events have affected the objective of the solution given by the last run of the solver ($T_{objective}$), yields slightly worse solutions but with a significant reduction in the number of reschedules necessary. Finally requiring an accumulated change in the process times of 100 time units provides reasonable results at even fewer reschedules.

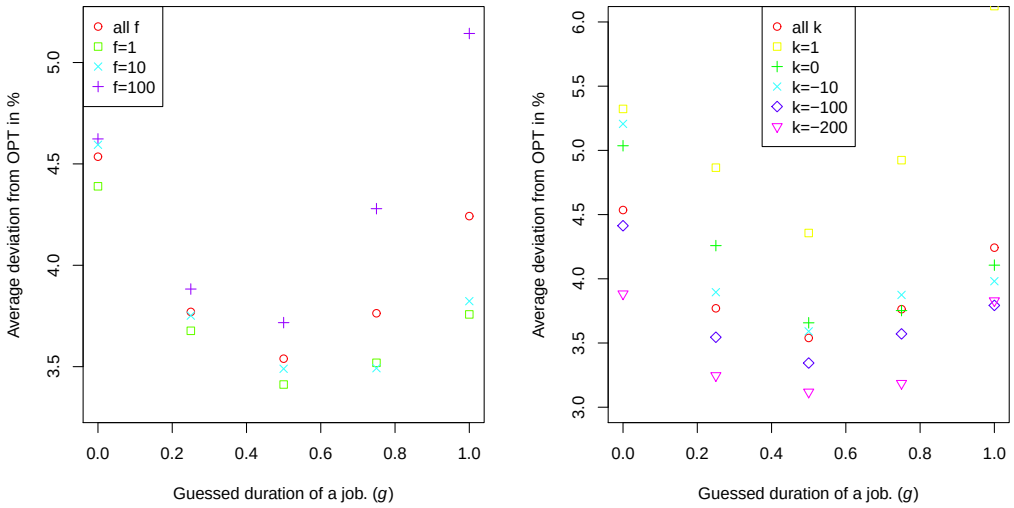

Figure 4.17: Solution quality as a function of the guess parameter g for JSP.

	$T_{objective}$	$T_{frequency}$	$T_{change10}$	$T_{change100}$
Avg. Reschedules	26.1	73.4	50.0	14.5
Average deviation	3.78%	3.75%	3.75%	3.95%

Table 4.6: The average number of reschedules and the solution quality when applying the four triggers $T_{objective}$, $T_{frequency}$, $T_{change10}$ and $T_{change100}$.

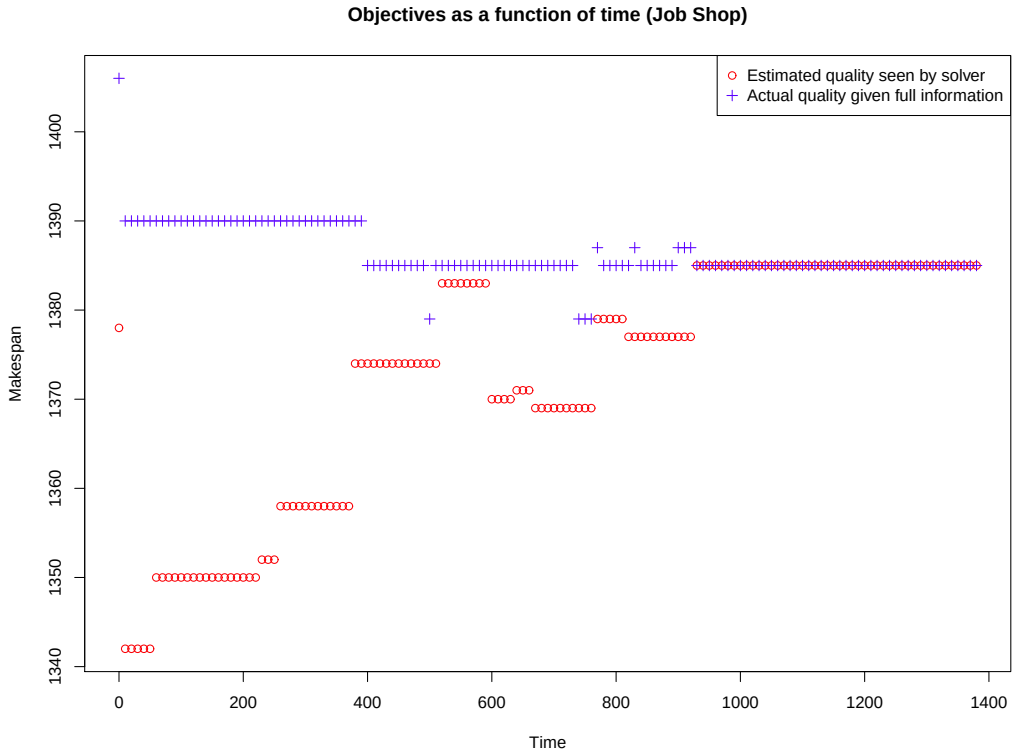

Figure 4.18: Progression of the objective function for the job shop problem with $extend \in \{0.5, 0.75, 1.0\}, k = -400, f = 10, g = 0.5$. Jumps in either graph could be due to a new solution being considered, but the estimated quality can change due to events changing distributions.

During the execution, a supervisor can monitor the stochastic Gantt chart, or the development of the objective function as shown in Figure 4.18. In the plot the estimated solution quality is the makespan as computed by the deterministic solver (red). Recall that the solver ignores uncertainties and the actual solution quality as computed by the simulator module reflects the real makespan (blue). The gap between the two typically starts out being large and tends to decrease over time as the schedule is executed and uncertainties are revealed. At the end of the execution the two values must coincide. In figure 4.18 the attained makespan value is 1385. Note that the actual solution quality generally improves over time, but that a solution with a better makespan (1379) than the final was obtained four times, but discarded due to lacking information. In a real life application, the full information is not known in advance and thus cannot be plotted. In this case another scenario (e.g. worst case, best case) can be monitored instead.

4.6.2.2 Test Case Analysis

For this test case we assume that $extend \in \{0.5, 0.75, 1.0\}$, $k = -100$ and $f = 10$ but with the option of either setting $k = -200$ or $f = 1$ at a cost.

	$T_{objective}$	$T_{frequency}$	$T_{change10}$	$T_{change100}$
Avg. Reschedules	26.2	58.9	44.8	14.5

Table 4.7: The average number of reschedules when applying the four triggers $T_{objective}$, $T_{frequency}$, $T_{change10}$ and $T_{change100}$ for $extend \in \{0.5, 0.75, 1.0\}$, $f = 10$ and $k \in \{-100, -200\}$.

In figure 4.19, the red circles correspond to maintaining f and k, while the green triangles and the blue crosses correspond to lowering f and k respectively. It is clear that getting information earlier by setting $k = -200$ is worth more on average than allowing rescheduling in each time step ($f = 1$), independently of the trigger utilized. The general quality of the solution appears to have no clear correlation with the trigger used, which further substantiates the hypothesis that the rescheduling is done often enough to take advantage of the available information.

Given that the triggers appear to give solutions of comparative quality, a decision maker would likely choose $T_{objective}$ or $T_{change100}$ to minimize the number of reschedules performed (table 4.7). For both of these triggers, setting $g = 0.5$ and $k = -200$ yields the best results: 2.96% and 3.07% respectively. To calculate if lowering the value of k is worth it, the benefit of 0.29% and 0.39% for $T_{objective}$ and $T_{change100}$ respectively must be compared to any exogenous costs of going from $k = -100$ to $k = -200$.

4.6.3 Comments

The application of the proposed dynamic rescheduling framework to two well known scheduling problems leads to interestingly conclusions, highlighting similarities and

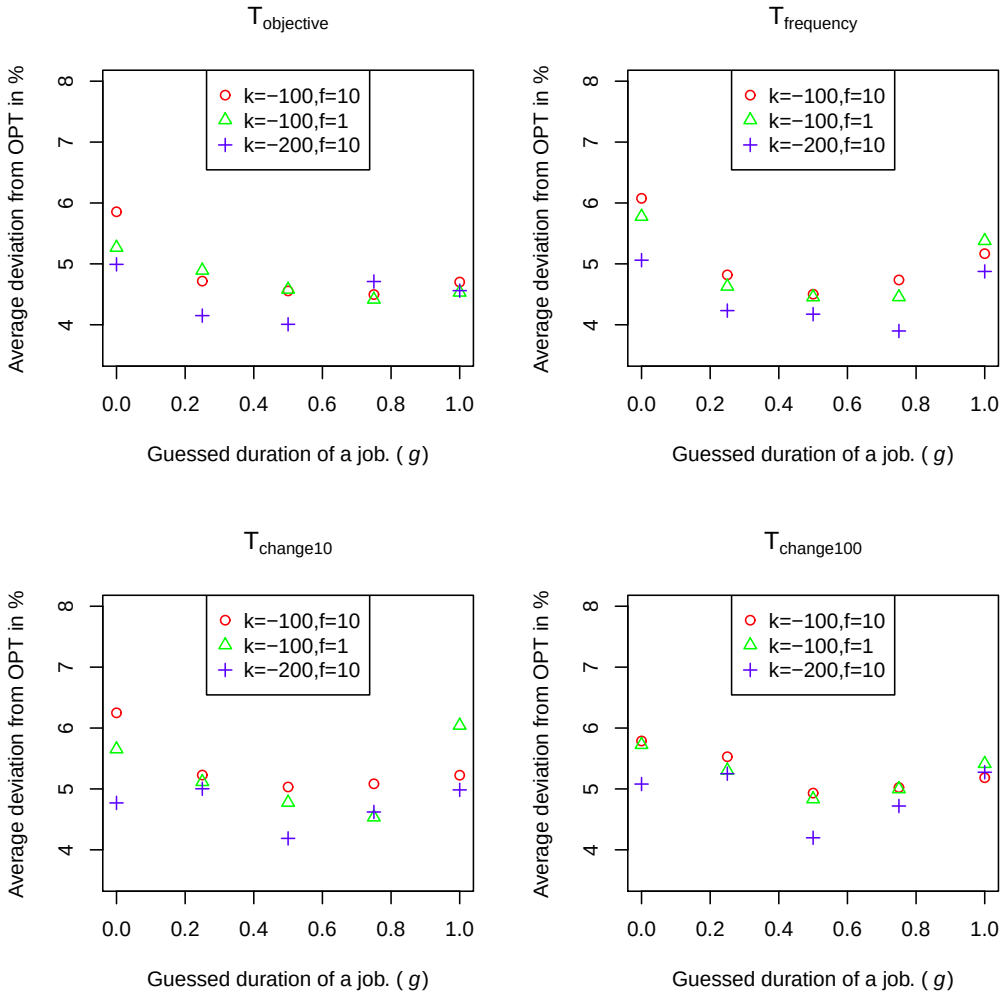

Figure 4.19: Average deviation from optimal ex-post solution as a function of g for each trigger for $extend \in \{0.5, 0.75, 1.0\}$.

differences in the behaviour of the two problems.

Apparently, the variation in quality in the specific test case for both the SMWCTP and JSP is mostly due to knowledge parameter, while the allowed frequency of rescheduling is less important. Having information earlier is more important than frequent reschedules.[6] However, it can be noted that the rescheduling frequency in the form of triggers has a more significant impact on the SMWCTP test case, that on the JSP test case.

For the JSP the most important factor turns out to be the magnitude of the uncertainties. When the problem allows only small changes (less than 20%) in the processing times there is no need of dynamic rescheduling since the ex ante solutions are already good enough. As the uncertainty increases the need of allowing rescheduling increases too.

When coming to the influence of the parameter setting of the framework, we observe a different behaviour for the two problems. In the Single Machine Weighted Completion Time Problem the choice of the trigger is influential, and while giving different results the sampling strategies do not radically change the quality of the solutions if $f \neq 1$. Whereas in the Job Shop Problem the choice of the trigger does not seem to have a strong influence on the solution quality while the number of reschedules changes a lot. When considering the sampling strategy g the best results are obtained by setting it to the expected (median) values of the processing times.

4.7 Possible Applications of the Framework

In this section we introduce possible applications of the framework, and discuss how they could be achieved.

4.7.1 Real Life Interface

The framework was designed with the primary purpose of being applicable in a real life production environment. The master thesis [Larsen 2008] describes a real life problem previously investigated by the author, which is compatible with the framework. The problem described in the master thesis [Skov 2007] also meets the criteria.

In each of these cases a decision maker is present, and would be able to engage in an interactive optimization approach. In [Larsen 2008] a rolling horizon approach defines a blocking job shop instance that is solved using the alternative graph model. Solvers used in Chapter 5 could be used without requiring much change in the developed Solution Reader and Instance Writer.

The solution proposed in [Skov 2007] also uses an alternative graph model, and could be integrated using the same solver. The problem is also a Job shop problem, but with a lot of 'no wait' constraints making the problem more challenging.

[6] When comparing in the relative magnitudes done in this chapter.

4.7.2 Utilizing CPU Capacity

The aim of both preemptive optimization, alternative optimization and 'combining solvers' is to utilize surplus CPU time to generate more robust solutions. While these methods require surplus CPU time, they can utilize multiple simultaneous threads, even if the used solver is unable to.

Alternative optimization is done by generating alternative deterministic instances for the solver to optimize. Generation methods can include worst case scenarios, best case scenarios, random scenarios, parametrized scenarios such as the one applied in this chapter, and finally scenarios generated based on analysis of the already obtained solutions. Examples of the last includes detecting critical paths, and overestimating durations on these in an attempt to force the solver to create solutions robust to changes in these.

Each of the solutions are analysed for robustness, and the solution having the most desirable properties can be selected and used. These decisions can be postponed until the time progresses past a point where the frozen part of the solutions considered would differ.

Preemptive optimization seeks to prepare for the outcome of future upcoming events. Instances can be generated base on possible scenarios for the next event, and when the event happens, the best of the candidate solutions can be selected.

Combining solvers runs multiple solvers (or solver configurations) instead of (or in addition to) generating different instances as described above.

The framework is capable of performing these three types of optimization, though generation methods and solution comparisons would have to be defined for all three methods.

4.7.3 Minimizing Deviation From a Schedule

In many applications the aim during the planning phase differs from the aim in the execution phase. This is typical of applications where a part of the schedule is publicly available and alterations is the target of minimizations. These cases includes train scheduling, airplane scheduling at airports and bus scheduling. The objectives in these cases are typically minimizing lateness while release times are enforced to eliminate early departures.

The framework is capable of handling these cases, and has been shown to work on the case from Chapter 5 in a preliminary study.

4.7.4 Comparing Solvers

Aside from using multiple solvers (or solver configurations) to generate solutions from which the "best" is chosen, the framework can also compare such solutions on a range of different scenarios, either tailor made for the comparison, or generated as Monte Carlo trials on the distributions. This comparison is made more efficient by applying the same Monte Carlo trials to each solution.

This application is investigated in Chapter 5.

4.7.5 Robustness and Sensitivity Analysis

Both robustness and sensitivity analysis can be performed without rescheduling, and the instance writer is thus not necessary for this purpose.

Robustness analysis takes one or more solutions to a scheduling problems and attempts to asses the effects of stochasticity on the solution(s).

This application is investigated in Chapter 5.

4.7.6 Evaluating Proposed Layouts

A problem often encountered in the industry is evaluating different designs for constructing work environments, machine shops and other facilities that will impose constraint on or define scheduling problems vital the the functionality of the company. Examples of such design decisions are:

- How much extra capacity should be added at a certain machine? This corresponds to adding workers to picking stations in [Larsen 2008], buying new machines in a scheduling environment with parallel machines and upgrading older/smaller machines.
- How many extra orders can be taken, and how would that affect deadlines?

The consequence of such decisions can be evaluated and visualized using the framework. Hence potentially leading to big savings compared to first building the facility and then having to change it due to bad performance.

4.8 Conclusions

In this chapter we have implemented a simulation-based framework for addressing dynamic rescheduling problems. The framework has capabilities similar to other frameworks found in the literature, but to our knowledge it is the only framework that features all of these properties simultaneously:

- Swappable and multiple solvers during the rescheduling process.
- Swappable and multiple objective functions evaluated during the rescheduling process.
- Storing multiple solutions.
- Storing multiple scenarios for comparisons of solvers or solver parameters.
- Decision support through generated visual representations.
- Event driven with the possibility of interfacing directly with a real life application.

The functionality of this framework has been shown by considering two different dynamic scheduling problems: A single machine and a job shop scheduling problem. Further examples of the provided functionality will be given in Chapter 5.

The framework has in this chapter been used to investigate some issues: *(i)* comparing the relative performance of different solver configurations and the absolute performance with ex-ante/ex-post optimization; *(ii)* evaluating the robustness of a solution and the sensitivity to uncertain distributions; *(iii)* the influence of triggers on the number of times rescheduling is performed, and the effect on the solution quality.

Future developments include the application of this framework to specific problems (such as trains scheduling or aircrafts landings and departures) and use of spare CPU time of the Solver to perform alternative future event branch prediction. A closed loop rescheduling paper for the train scheduling problem in Chapter 5 is planned.

Chapter 5

Robustness of Train Schedules

Contents

5.1 Contribution of the Author

The work documented in this chapter is based on a draft paper "Robustness of Optimal Train Schedules to Stochastic Disturbances of Process Times" written in collaboration with Andrea D'Ariano, Francesco Corman, Dario Pacciarelli and Marco Pranzo.

The solvers have been provided by the co-authors, but the tests, simulation and most conclusions are done by the author.

5.2 Introduction

Railway services are normally managed on the basis of a timetable (detailed plan of operations) that has been planned in advance with suitable time margins to absorb minor delays. However, during operations major disturbances cause traffic deviation from the planned timetable and thus may influence the timetable feasibility. In such cases, real-time adjustments of train timing and orders are required to assure compatibility with the real traffic situation and to limit delay propagation. This

task is currently performed manually by dispatchers. Recently, advanced models and algorithms have been developed to reschedule trains in complicated railway areas with dense traffic and multiple delayed trains (see, e.g., the literature reviews in [Corman 2011a, Hansen 2008, Lusby 2011]).

In rescheduling of railway operations, the need to quickly find solutions has directed most efforts to develop advanced heuristic methods, that find a good solutions with a limited computation effort. This is in line with a general trend for scheduling and managing systems under uncertainty for complex real life scheduling problems, that use solving procedures based on fully deterministic information [Ouelhadj 2009, Larsen 2012].

Such approaches simplify the problems by feeding the scheduling routines with only deterministic and static data and neglecting further uncertainties. However, most real life scheduling problems, and especially railway traffic scheduling, are inherently dynamic and prone to uncertainties. A key question is therefore whether the quality of the optimized solutions is partially or completely lost when dealing with uncertainty.

Assuming deterministic and static data may cause discrepancies between what is expected to happen in the model and what actually happens when the plan is applied in practice. In fact, the gap between the offline published plan and the actual schedule put into operations affect the operations' performance and railway passengers' comfort heavily.

Research aiming to increase the robustness of railway systems is generally directed at addressing uncertainty to fill this gap, mostly following two directions:

(i) computing schedules with some sort of slack to recover from disturbances,

(ii) assessing the effects of uncertainties of a pre-defined schedule before implementation.

In (i), optimal schedules can be computed using detailed models and dedicated algorithms, able to take the uncertainties and the dynamic nature of the problem into account. However such methods require the development of complex methods such as robust optimization [Ben-Tal 2002] or stochastic programming [Ruszczynski 2003] approaches which are generally harder to solve. The approach in (ii) is to use simple models able to compute efficient schedules based on deterministic information and to evaluate the effects of uncertainty, e.g., with the use of Monte Carlo samplings. This enables assessing the effects of stochastic disturbances on a schedule by running a simulation of the future evolution of the railway system.

In this chapter we follow the (ii) approach. Our aim is to develop a tool able to assess the effects of unpredictable disturbances on a schedule enabling a more informed evaluation of different rescheduling algorithms. In fact when dealing with real life problems one should also include the possibility that the system might evolve in directions unknown at the moment of optimization, and not considered in the schedule generated by the solver. This work studies the variability of optimized train schedules in the presence of uncertainty of both train travel time and their

dwell time at stations. A simulation setup is proposed, in which train schedules are computed minimizing delays in case of deterministically known perturbed operations. Such train schedules are computed by a microscopic optimization-based train scheduler, working at the level of block signals and we compute the process times in seconds. This detail is required to assess the propagation of delay between following trains precisely. The resulting solutions are then evaluated under small stochastic variations in process times, that simulate errors in input data due to uncertainty in train tracking, or further unpredictable events. This is done with the framework presented in chapter 4, that allows quantifying the effects of such disturbances precisely.

The chapter is organized as follows: Section 5.3 briefly reviews the train scheduling and robustness analysis literature. Section 5.4 presents the application of the framework for dynamic rescheduling for robustness evaluation while Section 5.5 describes the computational results by evaluating the robustness of three rescheduling algorithms on a complex dispatching area of the Dutch railway network. Finally conclusions and future research directions are highlighted in Section 5.6.

5.3 Literature on Robust Railway Systems

A growing stream of scientific literature focuses on addressing how to generate robust timetables. Even if there is still no clear consensus on the exact definition of robustness, as also discussed in Dewilde et al. [Dewilde 2011].

In fact several definitions have been proposed in the literature [Carey 1999, Fischetti 2009a, Liebchen 2009, Salido 2008].

Among the papers dealing with robust timetabling problems we distinguish between approaches developing optimization-based timetables with some degree of disturbance robustness, i.e., point (i) of the Introduction, and papers dealing with the assessment of the robustness of a timetable by means of simulation tools, i.e., point (ii) of the Introduction.

Liebchen et al. [Liebchen 2010] use an integer programming approach to generate a delay resistant periodic timetable and sequentially evaluate it according to a delay management strategy. The delay management problem [Corman 2012, Ginkel 2007] is to decide which connections should be kept in the actual plan and which should be dropped. Results are presented on a part of the German railway network.

Fischetti et al. [Fischetti 2009b] propose a light robustness approach to take robustness of a solution into account and at the same time maintain low computational times. On a medium sized network, the proposed approach results in two order of magnitude improvements for CPU time while maintaining almost the same solution quality.

Schöbel and Kratz [Schöbel 2009] present a bi-objective approach for generating robust and efficient timetables. Given a level of uncertainty, the problem of finding a Pareto optimal timetable is solved as efficiently as the original single objective

problem. However, it is generally difficult to predict the impact of the level of stochasticity of railway systems.

Shafia et al. [Shafia 2012] describe a mathematical model for train timetabling with some degree of robustness. As described by the same authors in their remarks, the model makes strong assumptions on safety issues related to train movements in the network. Based on this model and on two methods for the computation of buffer times, a branch and bound algorithm and a beam search-based heuristic are proposed by the authors for the computation of timetables. A real railway line is used for testing the algorithms while fictitious data are assumed for the travelling trains. The results show some improvements obtained by the two dedicated algorithms compared with a commercial software.

Vansteenwegen and Van Oudheusden [Vansteenwegen 2006] focus on the development of a robust timetable. They propose a two phase approach, in which, first, ideal buffer times with respect to train connections are computed, and then an LP is used to build a timetable minimizing the waiting costs. Finally a discrete event simulation is used to compare the original timetables with the new one. The authors reports substantial improvements.

Simulation is a common approach to evaluate the robustness of train schedule, as in point (*ii*) of the Introduction. And we comment on some railway applications of simulation tools to evaluate timetable robustness:

Carey and Carville [Carey 2000] compares the reliability of proposed schedules by simulating detailed train traffic in a busy complex station. The simulator uses a rescheduling algorithm to solve arising conflicts. Their approach can also be used to evaluate different reliability measures on a given timetable.

Salido et al. [Salido 2008] propose two analytical measures of robustness of a timetable and use a macroscopic simulator to validate the analytical and the observed empirical robustness. They use their approach on a case study with about 30 trains to compare two timetables.

Takeuchi et al. [Takeuchi 2007] propose a robustness index, measuring the passenger disutility introduced by a disturbance. They simulate the traffic on a line with 39 stations and 63 trains, to show the differences between two timetables.

To the best of our knowledge, very limited research has addressed the topic of robustness during rescheduling of railway networks, with a microscopic detail, and solutions based on optimization algorithms. Only recently Meng and Zhou [Meng 2011] developed a stochastic programming approach with recourse to incorporate different probabilistic scenarios in the rolling horizon decision, to reschedule trains in a perturbed situation, in order to minimize delays under different forecasted scenarios. Experiments on a single track line located in China with 18 stations and 50 daily circulating trains are carried out using a rolling horizon approach.

However there is no study trying to assess the performance of different rescheduling actions in real-time. Different train rescheduling plans can be obtained by using advanced conflict resolution methods, solving train conflicts and predicting the delay propagation in the overall studied area or adopting local and myopic rescheduling rules. On the one hand, an optimization algorithm based on deterministic infor-

mation may produce a more effective but fragile solution, i.e., an apparently good solution at the time it is computed, that is highly sensitive to uncertain factors, and that might result in unattractive plans. On the other hand, a rule-based algorithm may produce a seemingly less optimized but more robust solution, i.e., it may be able to absorb disturbances without a relevant degradation of the quality of the solution. However, in both cases, the impact of stochastic disturbances need to be carefully evaluated before schedule implementation.

The main motivation of our study is thus to quantify in real-time the improvement, if any, achievable by using optimization-based rescheduling methods under stochastic disturbances.

5.4 Robustness Evaluation Framework

Figure 5.1 presents the flow of data in the framework for dynamic rescheduling. And how it is used to evaluate the quality of train schedules when dealing with uncertain information.

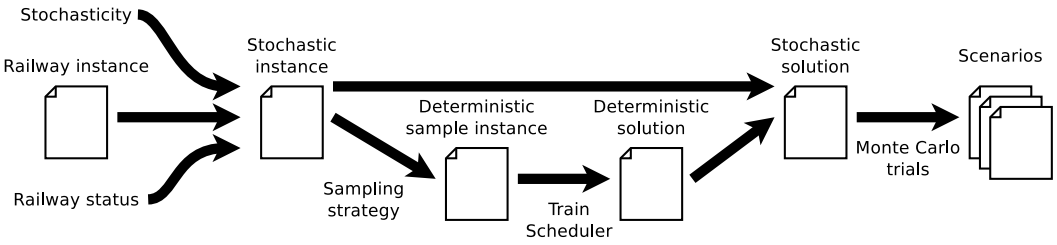

Figure 5.1: Flow of information in the framework.

The framework takes as input:

A railway instance. A deterministic railway instance contains a detailed description (at block section level) of the railway network under study, the set of trains running in the network over the considered time horizon and the timetable they should follow on a railway network.

Railway network status. The current status of the network, i.e., train position and speed of all trains at time t_0, is assumed to be known. Based on this information it is possible to generate the actual entrance delays in the network that must be provided to the solver. The entrance delays are generated as deterministic values of train positions and speeds, that represent the current status of the network and the expected evolution in the short term.

Stochasticity information. The simulator needs to know the probability functions of the duration of all operations subject to uncertainty. If an operation is not deterministic the probability function should be known and specified. In our tests we considered uncertain duration for all running and dwell times.

All this input defines a stochastic instance. Since it is assumed that the train scheduling solver is not able to directly tackle a stochastic train scheduling instance the framework must transform it into a deterministic instance.[1] This is done as follows: The probability function associated to each operation (running times and dwell times) are converted into a deterministic value by means of a sampling strategy. As a sampling strategy one could consider taking the median of the probability function, the expected value, or a given quantile.

Next the train scheduler is invoked on the deterministic instance and it is expected to produce a deterministic solution to the problem. The scheduler uses a train scheduling algorithm in order to detect and solve potential conflicts, and outputs a microscopic plan of operations, with a level of detail of single block sections and a precision of seconds.

We use ROMA [D'Ariano 2009, D'Ariano 2008] as solver: ROMA is a computerized system that can assist train operators in their tasks. It is able to estimate and control the future evolution of the railway traffic considering deterministic information on track occupation and safety constraints, as well as train dynamics. This may prevent the decision maker from taking less effective decisions, such as causing a deadlock situation or unsatisfactory throughput. The resulting solution represents a new plan of arrival and departure times that minimizes train delays with respect to an original timetable. ROMA also enables the optimization of railway traffic if the actual timetable is not conflict-free and/or deadlock-free, or during severe traffic disturbances, such as when emergency timetables are required and dispatchers need extensive support to solve train conflicts and to handle disruptions (i.e., unexpected blockage of some tracks).

After the computation of a new schedule by the solver, the framework replaces deterministic process times in the deterministic solution with their original probability functions from the stochastic instance, creating a stochastic solution.

A set of Monte Carlo trials are then performed on the stochastic solution creating a set of sample solutions (scenarios). On the basis of the generated scenarios, train delay indicators are computed.

Observe that the proposed framework does not add excessive computational burden on the solver. In fact, the more costly operation in the framework is to evaluate the robustness of a solution. For large scale instances (as the one we consider in our tests) this can be done quickly. Furthermore, different solvers can be used as long as they are able to handle a stochastic or a deterministic train scheduling instance and provide a solution with the desired level of modelling detail.

5.5 Case Study

In this section we first introduce the experimental setup of the tests and then we evaluate the robustness of different train scheduling algorithms.

[1] Clearly, if the solver is able to deal with a stochastic instance there is no need of removing stochastic data.

5.5.1 The Railway Network

The test case is a complex and densely occupied area of the Dutch railway network, as in [Corman 2011b]. We study the central station of Utrecht area, which is a major hub of the Dutch railway network, at the crossroads of the five main lines criss-crossing the Netherlands. About 200.000 passengers use the station every day. More specifically, there are:

- a double track line to the west, towards Rotterdam and The Hague, served by 12 trains per hour per direction;
- a four-track line to the north-west, towards Amsterdam, served by 11 train per hour per direction;
- a four-track line to the north, towards Amersfoort, served by 12 trains per hour per direction;
- a double track line to the east, in the direction of Arnhem and Germany, served by 6 trains per hour per direction;
- and finally a double track line to the south, towards Den Bosch, served by 9 trains per hour per direction.

The train network under consideration is shown in Figure 5.2. The lines are interconnected by two complex interlocking areas at both sides of the station with a total of about 100 switches, and 20 platform tracks at the station itself. The area under consideration stretches along a few kilometres of the lines, for a diameter of about 20 km.

5.5.2 Timetable and Deviations

The 2008 timetable is considered, which is very dense, featuring 80 trains per hour on a regular-interval timetable. The traffic is mixed, divided approximately evenly between intercity trains and commuter trains, with a few long-distance, high-speed trains. We consider two train scheduling instances, with a time horizon of traffic prediction of 3600 and 7200 seconds. Those two instances correspond to a set of 1962 and 3924 stochastic process times respectively, corresponding to travel and dwell times subject to delays.

For each of the two instances we assess 40 cases of entrance delays from [Corman 2011b] representing delays that have already happened before the time window under consideration in the instances, or are expected in the short term. Each train has a *planned arrival time* associated with each scheduled stop, giving rise to different delay measures.

The difference between plan and realization is the *total delay* of a train at a scheduled stop (equivalent to a *lateness* measure), and can be further divided in two parts: *Primary delays* that cannot be avoided even by letting train running at maximum allowed speed and with no hindrances, and *consecutive delays* (or *secondary delays*)

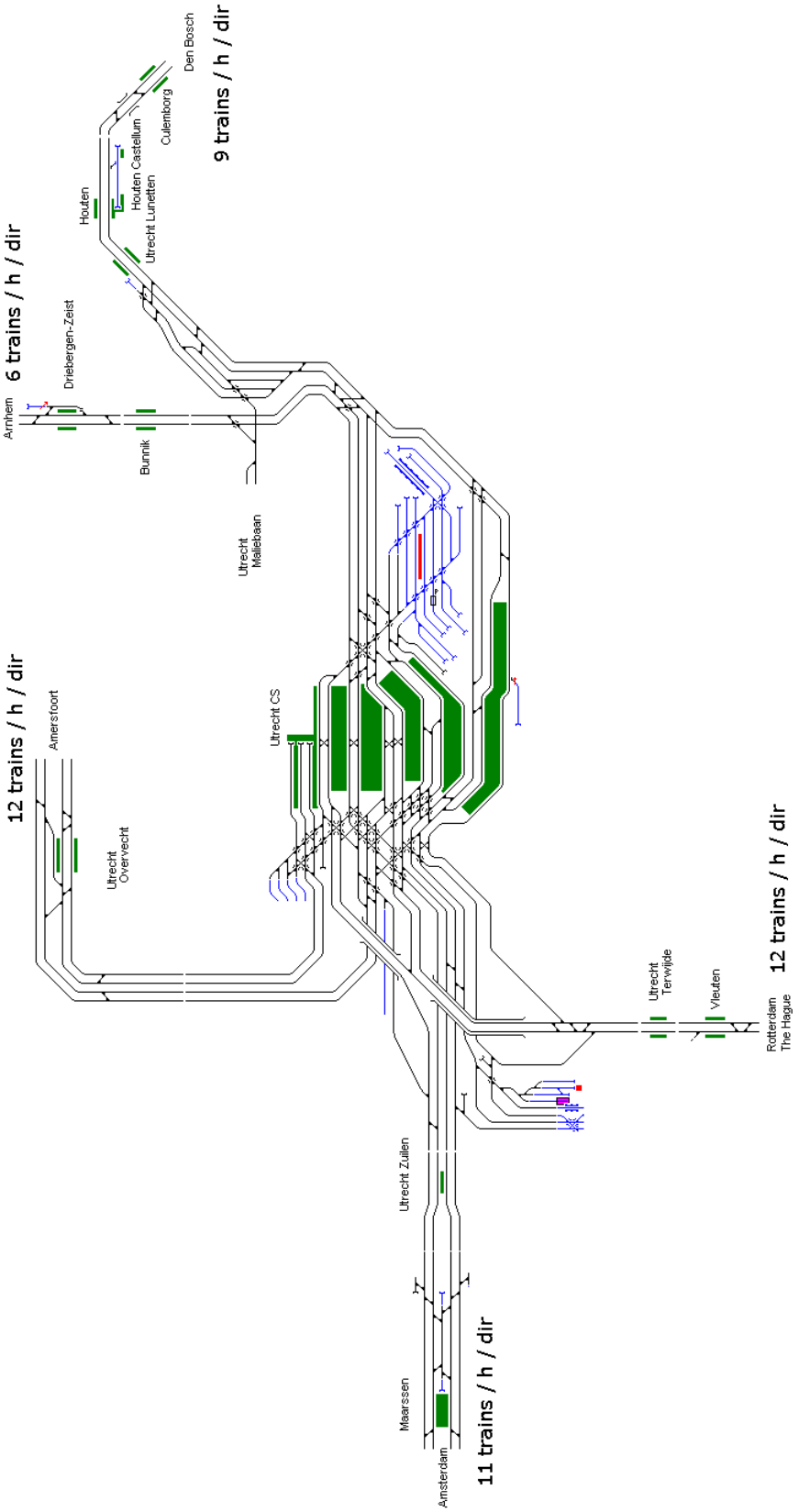

Figure 5.2: Map of the Utrecht area.

which are caused by domino effects and usually can be partially avoided through a careful planning of the operations [Corman 2011a, Corman 2011b]. Since primary delays are by definition unavoidable, we use the minimization of consecutive delays as performance measures (that is the output delay due to potential train conflicts in the area under study). More specifically, in our test, we show the minimization of the maximum and average consecutive delays.

The schedule robustness is evaluated by the analysis of small stochastic variations for the running and dwell times of train traffic in the network. Those variations can be the result of e.g., measurements errors, coarse grained train detection data (for instance based on block section occupancy, rather than a precise positioning system), additional unexpected disturbances, overcrowding or additional delays at station platforms.

We generate 1000 random instantiations of the associated probability functions and use them to evaluate schedule robustness. In other words, each stochastic solution is evaluated over 1000 Monte Carlo trials (scenarios). Observe that, in order to obtain a fair comparison, when evaluating the robustness of the solutions provided by different algorithms on the same instance, we apply exactly the same scenarios in the Monte Carlo trials phase.

Since the scheduling solvers are deterministic all durations must be supplied as deterministic values and not as probability functions. The proposed robustness evaluation framework has a single parameter that has to be set. Namely we have to specify how the sampling strategy works, i.e., how a stochastic instance is transformed in a deterministic instance. To this aim we use a sampling strategy parametrized on:

$$g \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$$

Those values represent the quantiles of the probability functions given in the stochastic instance, ranging from $g = 0.0$ to $g = 1.0$. Setting $g = 0$ means that the scheduling solver is considering an instance where all the times are set to the shortest possible duration allowed by the distribution of that operation. When $g = 0.5$ the sampling strategy provides the median times, and when $g = 1$ the maximum of each distribution is taken. As we use unbounded Weibull distributions, we treat $g \leq 0.99$ as $g = 0.99$ for them to avoid infinities. Note that for all the distributions used, the expected values of the distribution corresponds to using $g = 0.5$ (Uniform distributions) or $g \approx 0.56$ (the used Weibull distributions).

5.5.2.1 Computing Consecutive Delays

Recall that consecutive delays were defined as the part of the total delays that were unavoidable regardless of scheduling decisions, and note that operations are indexed by j. Arrival times t_j^a are computed as longest paths in the temporal network representing the solution $G = (N, F \cup S)$. Given a planned arrival time t_j^p, the lateness can be calculated as $t_j^l = max(t_j^p - t_j^a, 0)$. The unavoidable delay t_j^u is then computed as the longest path in the temporal network without any alternative

edges selected ($G = (N, F)$). Finally the consecutive delay can be computed as $t_j^c = t_j^l - max(t_j^p - t_j^u, 0)$ which is guaranteed to be positive or zero.

In the train scheduling problem we consider both maximum consecutive delay:

$$\max_{j \in planned}(t_j^c) \tag{5.1}$$

and average consecutive delay:

$$\frac{\sum_{i \in planned}(t_j^c)}{|planned|} \tag{5.2}$$

where *planned* is the set of operations with a planned arrival time and $|planned|$ is the sets size.

5.5.3 Solver Settings

The scheduler works with deterministic information based on an alternative graph formulation of the train scheduling problem. Three scheduling algorithms are considered: the simple dispatching rule First In First Out (FIFO), a greedy heuristic (AMCC) [Pranzo 2003] and the truncated Branch and Bound (B&B) algorithm [Corman 2011a]. The FIFO rule simply states that the train arriving first at a track junction or crossing point passes first. The AMCC heuristic is based on the alternative graph formulation of the train scheduling problem [D'Ariano 2008] and it basically takes sequential decisions trying to avoid the worst possible decisions at each time. The B&B is an advanced branch and bound algorithm using both FIFO and AMCC as initial solutions and a Jackson Preemptive Schedule lower bound. In order to guarantee reasonable computing times the B&B is truncated after 120 seconds.

The deterministic scheduling solution computed by the three algorithms is transformed into a stochastic solution using predefined probability functions, and then evaluated using 1000 instantiations of the stochasticity in the process times. From our tests it turns out that, given a solution, the analysis of the 1000 scenarios can be done in about 10 seconds for the largest instance and all objectives. This time is affected by the measures of consecutive delays, as they require an extra longest path calculation to be made to determine unavoidable delay. The process is easily parallelized if faster computation is needed.

5.5.4 Distribution of Process Times

We consider probability distributions for the running and dwell times as follows. Weibull distributions are used to characterize the variability of relevant process times, as in [Corman 2011b, Yuan 2006]. Specifically, the same dataset of events is used as in [Corman 2011b], that contains about 33000 records of arrival and departure events at Utrecht station, corresponding to a month of operations. When studying dwell times, we divided the events into *on peak* and *off peak* events. A

standard Maximum Likelihood fitting procedure is used to derive the three parameters describing the two Weibull distributions. Distributions of peak dwell times have a shape parameter $\kappa = 1.7914$, a scale parameter $\lambda = 255$ and a shift of -16 seconds towards shorter durations. The off peak dwell times have $\kappa = 2.0824$, $\lambda = 261$ and a shift of 4.

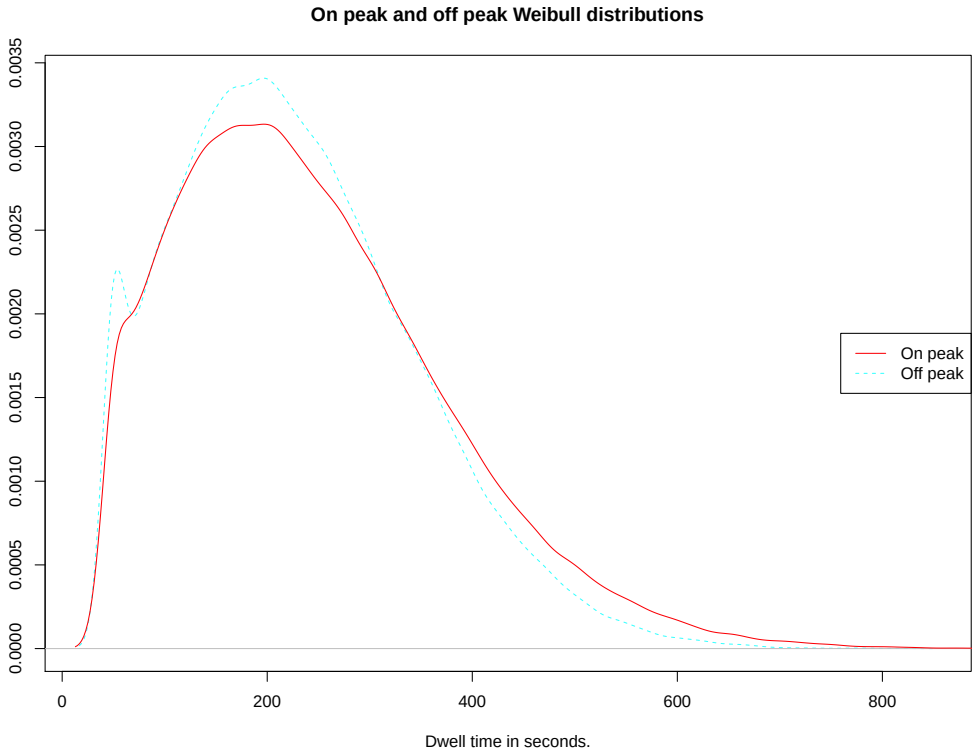

Figure 5.3: Densities of dwell times for trains at stations. The spike at time 48 represents the enforced minimum. The height of the curve indicates the likelihood of the probability distribution assuming that delay.

Densities for the two probability distributions (on and off peak dwell times) are plotted in Figure 5.3. As expected, during peak hours the larger amount of passengers results in longer dwell times. Both probability functions are bound on the lower values by a minimum technical stopping time of 48 seconds, that results in a small peak on the probability density graph.

Another source of uncertainty is related to running times. This is modelled as a variation of the running time per every single operation (i.e. the passage of a train over a block section); the variation is considered normalized with regard to the planned duration, and follows uniform distributions.

The probability functions related to different trains and different block sections have no correlation in time or space. Unfortunately, little research has been focused on the assessment of the variability of individual running times at a microscopic level. Further research would be required to have a comprehensive description of

the statistical characteristics of running time variations, based on recorded data.

We consider four different levels of distributions of running time variations: the intervals describing the uniform distributions are respectively $v = [-2.5\%, +5\%]$, $[-5\%, +10\%]$, $[-7.5\%, +15\%]$ and $[-10\%, +20\%]$, i.e. there is a slight bias towards longer durations.

5.5.5 Robustness Evaluation

Using two instances, four sets of process time stochasticity, on/off peak behaviour, 40 entrance delay cases and ten sampling strategies gives 6400 runs per algorithm tested. adding 1000 scenarios and a deterministic case, three algorithms and four objectives this gives 76.876.800 evaluated objectives. This was completed with 12 machines, with the last one finishing after 36 hours.[2]

The plots in Figure 5.4 shows the effect of the sampling strategy.[3] The plots are averaged over all cases and on peak/ off peak dwell times. The right plots shows the average and maximum consecutive delays obtained by the solvers, and the left plots are a result of applying these solutions to the 1000 scenarios. The right plots shows a gradual increase in objective values, which is expected as all process and dwell times increase as g grows. Note also that this growth is not representable of what would happen in a real life scenario, as the 1000 scenarios on the left show.

The missing values in Figure 5.4 indicates that the greedy heuristic failed to find a feasible solution for $g \in \{0.4, 0.7, 0.8, 0.9, 1.0\}$. It does however produce competitive results for $g \in \{0.5, 0.6\}$ faster than B&B.

The spread between the scheduling algorithms is in general limited, unless a large value of g is considered, representing a more conservative expectation of the process times. The closeness of results between algorithms for the same value of g, indicates that the g parameter is a stronger predictor of the quality of the stochastic solution than the selected algorithm. Note that underestimating delays when solving the deterministic instances corresponds to low values of g ($g \approx 0.33\ldots$), and thus result in worse solutions on average. Using expected values for the probability distributions (corresponding to $g = 0.5$ for the uniformly distributed running times, and $g \approx 0.56$ for the dwell times following Weibull distributions) yields much better results, and if the distributions of the delays are unknown, overestimation is better than underestimation according to Figure 5.4.

Observe how assuming $g = 1$ leads to poor solutions. This is due to the presence of unbounded distributions i.e. the Weibull used to model dwell times. More specifically, assuming $g = 1$ ($g = 0.99$) corresponds to sample extremely large dwell time thus shadowing the travel times and producing poor quality solutions when stochasticity has been applied.

A value of $g = 0.6$ yields the best solutions for all three algorithms, and will thus be used in subsequent comparisons between the three algorithms.

[2] It was a single machine running the last of the larger instances.

[3] The same plot split for on peak and an off peak, can be seen in appendix C figure C.1 and C.2.

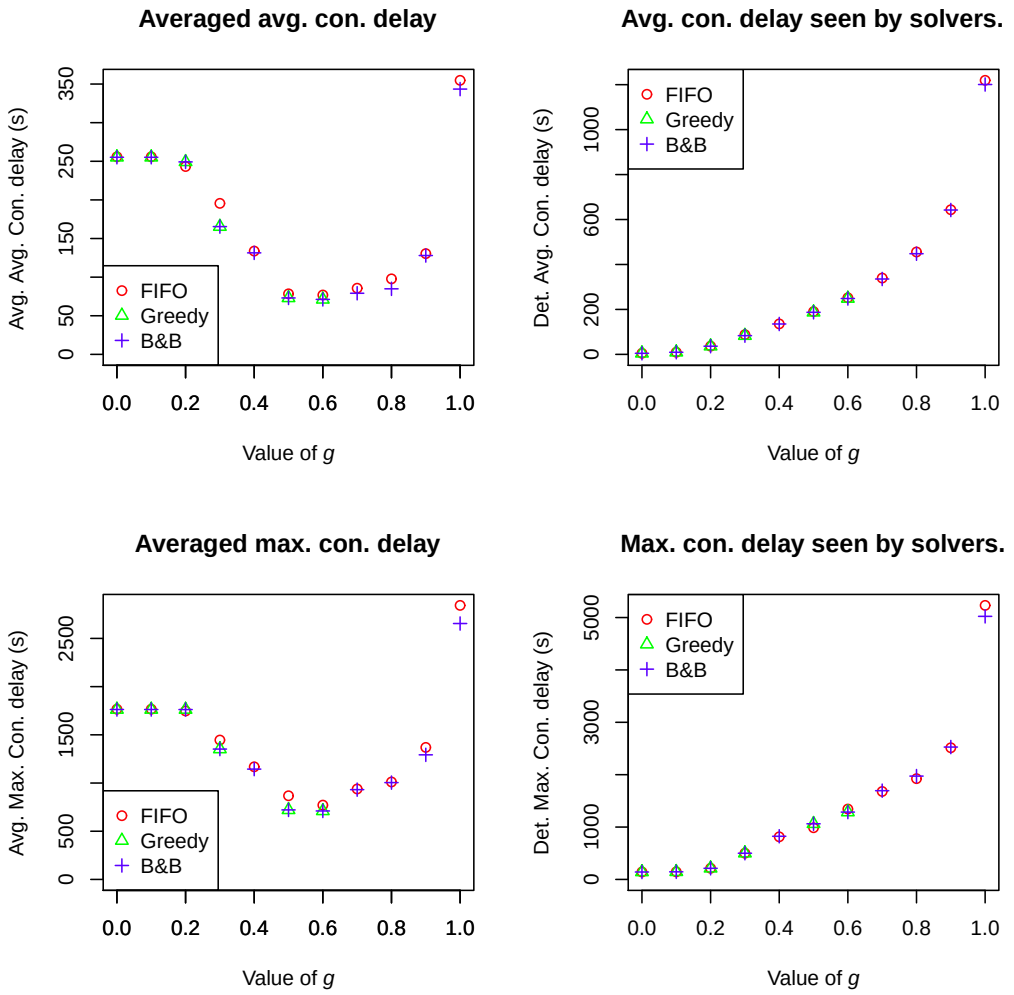

Figure 5.4: Average values of consecutive delays as a function of the sampling strategy (g). The two plots to the right is the solution quality produced by the solvers, and the two plots to the left is the quality after the Monte Carlo samplings are applied to the same solutions.

Off peak		Deterministic			Stochastic		
Stoch.	Obj. func.	FIFO	B&B	%	FIFO	B&B	%
$[-2.5\%, 5\%]$	Avg. Late.	223.02	219.1	1.76	241.01	235.76	2.18
	Max. Late.	1160.00	1160.00	0.00	1456.41	1460.19	-0.26
	Avg. Con.	27.36	23.44	14.34	63.75	58.50	8.23
	Max. Con.	256.50	253.00	1.36	667.89	658.09	1.47
$[-5\%, 10\%]$	Avg. Late.	233.54	229.74	1.63	247.53	242.09	2.20
	Max. Late.	1208.00	1208.00	0.00	1482.45	1482.15	0.02
	Avg. Con.	29.14	25.33	13.05	65.38	59.94	8.32
	Max. Con.	280.50	280.50	0.00	670.63	653.20	2.60
$[-7.5\%, 15\%]$	Avg. Late.	242.40	240.76	0.68	254.76	249.91	1.90
	Max. Late.	1252.00	1252.00	0.00	1508.07	1503.40	0.31
	Avg. Con.	30.46	28.81	5.39	67.79	62.94	7.15
	Max. Con.	305.50	303.00	0.82	676.21	652.42	3.52
$[-10\%, 20\%]$	Avg. Late.	255.78	254.48	0.51	261.58	257.39	1.60
	Max. Late.	1456.50	1304.00	10.47	1603.30	1531.23	4.50
	Avg. Con.	33.10	31.81	3.91	69.39	65.20	6.04
	Max. Con.	361.50	331.00	8.44	736.55	659.19	10.50

On peak		Deterministic			Stochastic		
Stoch.	Obj. func.	FIFO	B&B	%	FIFO	B&B	%
$[-2.5\%, 5\%]$	Avg. Late.	247.05	244.63	0.98	282.09	277.27	1.71
	Max. Late.	1238.00	1238.00	0.00	1640.27	1640.12	0.01
	Avg. Con.	32.89	30.47	7.35	82.74	77.91	5.83
	Max. Con.	325.50	297.00	8.76	777.00	754.66	2.87
$[-5\%, 10\%]$	Avg. Late.	257.88	256.05	0.71	288.70	284.28	1.53
	Max. Late.	1453.00	1286.00	11.49	1712.81	1664.68	2.81
	Avg. Con.	34.66	32.83	5.28	84.24	79.81	5.25
	Max. Con.	376.50	323.50	14.08	816.48	754.47	7.59
$[-7.5\%, 15\%]$	Avg. Late.	266.92	265.55	0.51	295.37	291.05	1.46
	Max. Late.	1477.50	1330	9.98	1735.24	1690.04	2.60
	Avg. Con.	36.13	34.76	3.79	86.08	81.77	5.01
	Max. Con.	362.00	348.50	3.73	817.38	760.29	6.98
$[-10\%, 20\%]$	Avg. Late.	287.29	277.97	3.25	310.29	298.01	3.96
	Max. Late.	1507.00	1507.00	0.00	1754.92	1753.14	0.10
	Avg. Con.	45.73	36.40	20.39	95.55	83.27	12.85
	Max. Con.	722.50	339.50	53.01	1006.18	789.95	21.49

Table 5.1: Average results on deterministic and stochastic instances for $g = 0.6$, split by off/on peak, process time stochasticity and objective function. The greedy heuristic has been omitted since we obtained the same solutions as B&B for $g = 0.6$. The % column indicates how many % FIFO was outperformed by B&B on average.

Table 5.1 documents the results of the algorithms on instances with different levels of stochasticity. The first half of the table is the off peak situation, and the lower half is the on peak situation. The columns are then grouped according the the stochasticity of the process times, and all four considered objectives are shown (in seconds). The columns are split in the deterministic case, which documents the output of the deterministic solver, and the stochastic case which apply the same solutions to the 1000 scenarios. The % columns report the percent wise difference in performance between B&B and FIFO.

When comparing deterministic and stochastic instances in Table 5.1, it must be noted that the train delays are much larger when stochasticity is considered, on average 10% for total delay, and up to $\approx 250\%$ for average consecutive delay. This reflects a high sensitivity of the train movements to delay propagation, and justifies the usage of microscopic models that can compute delay propagation between subsequent trains precisely. The ratio between average delays and maximum delays (both consecutive delay and lateness/total delay) remains mostly the same across the different stochastic levels, and algorithms. B&B generate consistently better solutions with respect to the maximum and average consecutive delay, both in the deterministic and the stochastic case.

Surprisingly the relative gap in average consecutive delay between B&B and FIFO actually increases in a significant part of the cases, when stochasticity is added. It was hypothesised that the tighter schedule would make more delays propagate. We also observe that as the maximum consecutive delays produced by the B&B remains almost constant as the stochasticity v increases, the solutions produced by the FIFO shows an increasing pattern for the maximum consecutive delay. The difference between algorithms increases slightly with the amount of stochasticity v, due to both a greater expected value of the process times, and a greater chance of delay propagation due to the larger variability.

The plots in Figure 5.5 give further details on the statistic trends, aggregating over all values of g, all stochasticity levels v and on/off peak. Also in this case the greedy heuristic has been omitted from the plots, as it fails to give a feasible solution in all cases thus skewing any density plots. The plots on the left report the distribution of the averaged average[4] lateness and the averaged average consecutive delay incurred by FIFO and B&B. Solutions computed by B&B are in general better, as observed in Table 5.1. This can be seen by a shift towards lower delays for both objective functions. Another interesting feature of B&B is the reduced spread compared to FIFO, i.e., its distributions has a sharper peak. This means that B&B is able to come up with good solutions more consistently and suffers from uncertainty to a lesser extent. Similar trends arise for the average lateness. Finally, the ripple pattern on the right tail of the figure is due to the aggregation of multiple values of g into a single plot. This highlights the role of the sampling strategy, that has a relatively large influence on the final solutions. The plot on the right represent the densities of the average maximum lateness and the average maximum

[4]When two aggregators are specified, the inner aggregates over the operations in the solution.

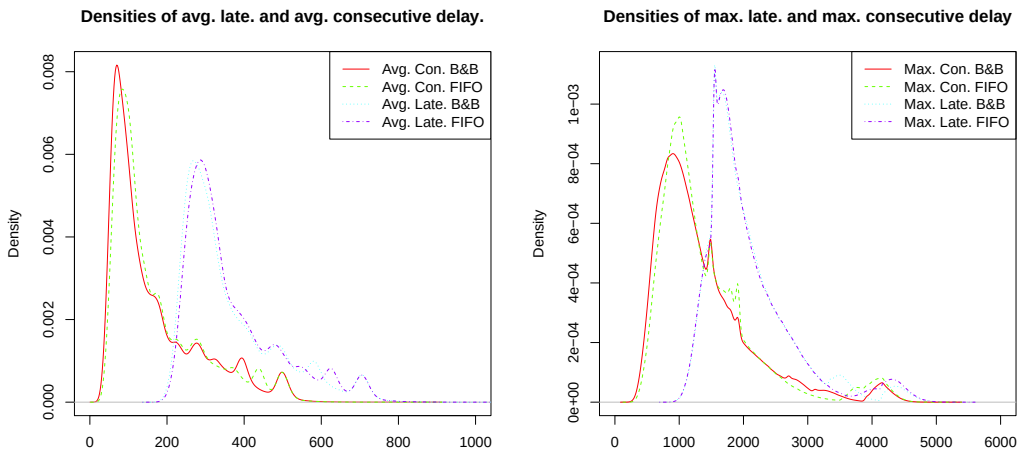

Figure 5.5: Densities of the average delays and consecutive delays (left) and maximum delays and consecutive delays (right) for FIFO and B&B in seconds, for all g, for all v.

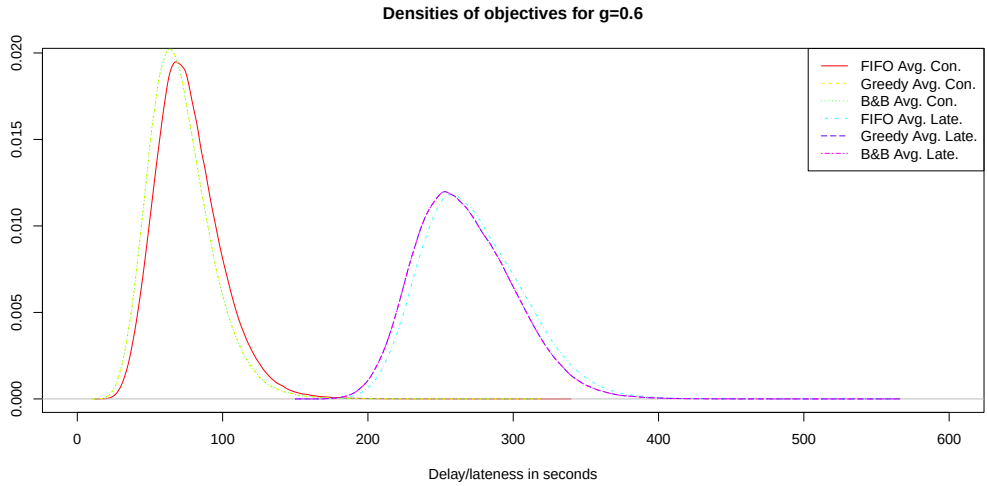

Figure 5.6: Densities of average consecutive delay and average lateness for FIFO, greedy and B&B when $g = 0.6$, for all stochastic levels v.

consecutive delay. B&B outperforms FIFO on average consecutive delays as seen in Table 5.1, while suffering from larger average maximum lateness values. In general, the same patterns arise as for the average delays, but due to the high non-linearity of the maximum delay as indicator, the plots are more erratic, and trends are more difficult to visualize.

Figure 5.6 shows the contribution to left plot in figure 5.5 that comes from the case where $g = 0.6$, for all stochasticity levels v. The right tail shows a very neat reduction towards zero; the other trends that were discussed in Figure 5.5 are more easily seen. For this set of experiments, the greedy heuristic and B&B are indistinguishable.

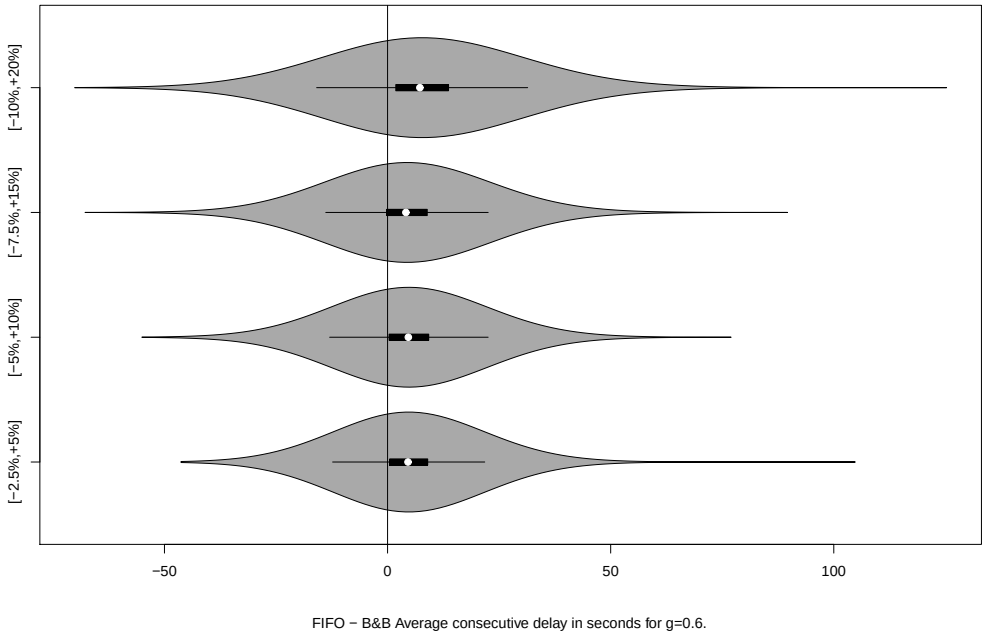

Figure 5.7: Difference between FIFO and B&B when $g = 0.6$, for all v. The area of shape left of the vertical line, indicates the cases where FIFO outperformed B&B after stochasticity is considered, and vice versa. The corresponding box plots are drawn in the shape.

Figure 5.7 illustrates the difference in average consecutive delays between solutions produced by FIFO and B&B for all Monte Carlo samples for $g = 0.6$, for all stochastic levels v. Since the plot is skewed to the right is clear that B&B is the best algorithm, on average. On the other hand, the tails of the distributions are quite large, so there exist samples for which the solution provided by the FIFO performs better than the one produced by the B&B on the scenarios. The FIFO solutions happen to be better after stochasticity is added in the 22.53% of the cases for $v = [-2.5\%, 5\%]$, in 23.35% for $v = [-5\%, 10\%]$, in 26.03% for $v = [-7.5\%, 15\%]$ and finally in the 17.77% of the cases for the higher stochastic level $v = [-10\%, 20\%]$.

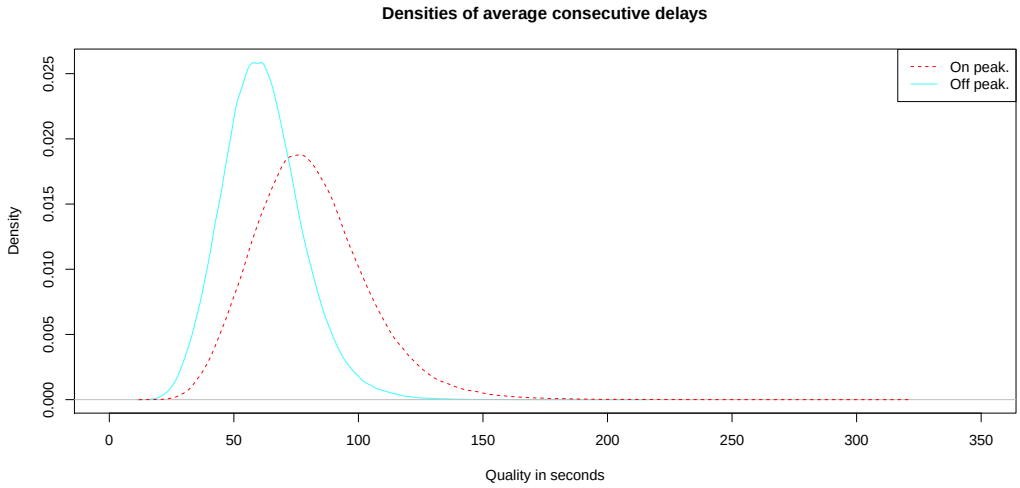

Figure 5.8: Densities of average consecutive delay produced by B&B for $g = 0.6$, for all v. The steeper shape of the density plot for the off peak case, indicates that the average consecutive delay tend to be smaller as expected.

Again, these results show that the B&B is able to produce nearly as robust solutions as FIFO and the gap between them increases as the travel times become more uncertain.

Two scenarios were considered for dwell time durations: on peak and off peak. Figure 5.8 demonstrates a shift in average solution quality: when dwell time increases in the peak hours (Figure 5.3), the resulting consecutive delay suffers quite a lot, with about 15 seconds deviation on average. Thus, a relatively limited difference in dwell time probability may result in remarkable differences in output. This confirms the general idea that stations are the main bottleneck of railway systems, specially for large stations such as the one considered in this study, that have many conflicting inbound and outbound routes, and serve large amounts of passengers.

5.5.6 Rescheduling Analysis

It can be observed that in practice, when the delays exceed a given threshold the human dispatcher may execute the rescheduling algorithm to produce a new plan more adherent to the current network status. Therefore an algorithm able to produce a robust schedule may lead to a reduced rescheduling frequency, which is desirable. The proposed robustness evaluation framework can then be used to evaluate how often the given rescheduling threshold is exceeded and the rescheduling process is triggered. To determine in what percentage of cases a rescheduling is required, we must first decide on the threshold level to consider. We used the average consecutive delay as objective function indicator, because it allows comparisons between instances with different primary (unavoidable) delays. It is also clear that the lateness measures are a poor basis on which to decide if a rescheduling is needed, as

all of them might be unavoidable. The percentage of simulated outcomes requiring rescheduling is plotted against the chosen threshold level based on the average consecutive delay in Figure 5.9. In the figure we show the behaviour of FIFO and B&B for a low stochasticity ($v = [-2.5\%, 5\%]$) and high stochasticity ($v = [-10\%, 20\%]$) situations when $g = 0.6$ is considered. From the plot it is clear that average values below 50 are rare, while values greater than 150 seldom occur. For values in between, there is a consistent gap between B&B and FIFO, indicating that choosing acceptable average consecutive delays in this interval would force more frequent reschedules for FIFO than for B&B.

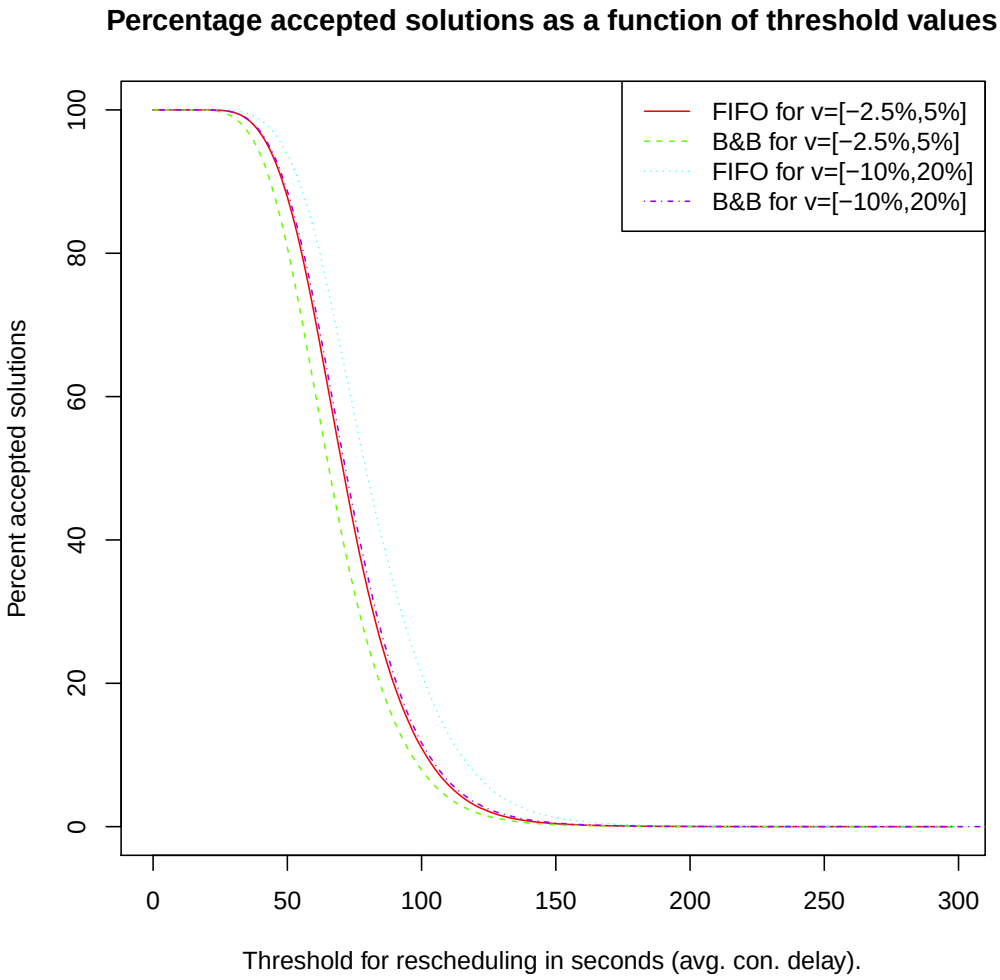

Figure 5.9: If an acceptable level of average consecutive delay is defined, the resulting percentage of scenarios requiring rescheduling can be determined. These values are computed for $g = 0.6$.

5.5.7 Analysing a Single Dwell or Process Times Influence

Another value of interest is the influence on the overall schedule of the variation of a single dwell operation. To this aim we have to identify the operation with the greatest potential impact on the average consecutive delay of all trains.

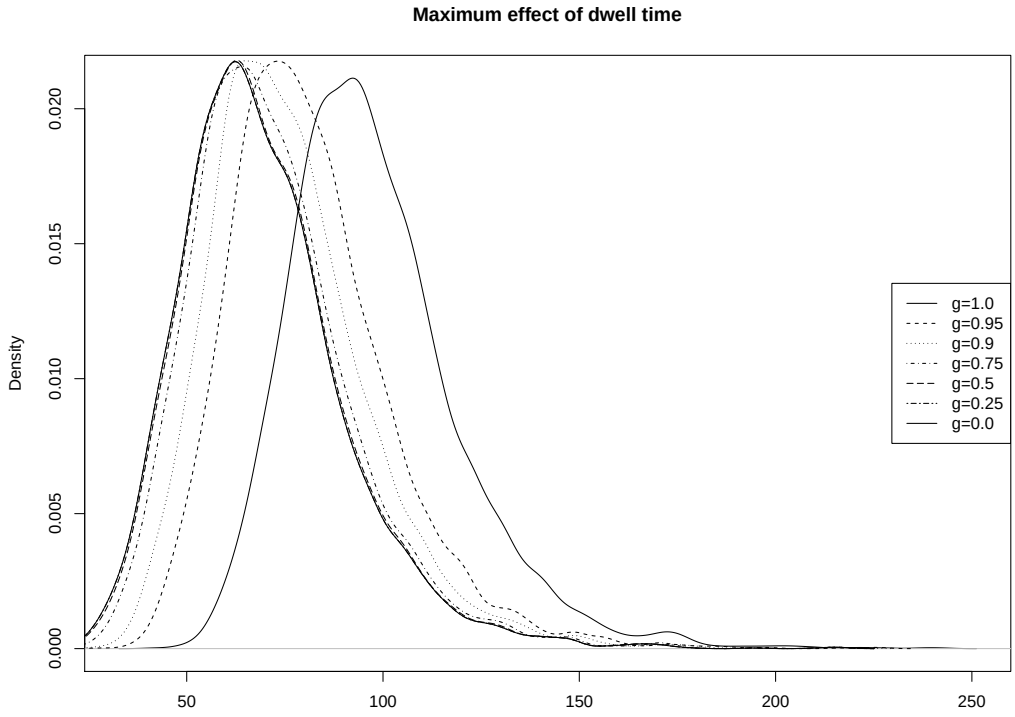

Figure 5.10: Average consecutive delays sensitivity to changes in a single dwell time on a 3600 seconds instance. The corresponding plot for trains travel times shows little discernible difference for $g = 0$ and $g = 1$, even with $v = [-10\%, +20\%]$

Longest path calculations done on the graph are used to obtain starting times for each operation, and allows for identification of critical paths. By selecting an operation o_i to analyse, changes can be made to that operations process times for all the sample graphs[5], and the resulting sets of objectives can be analysed.

The operation with the greatest potential impact on the objective is identified by applying the sampling strategies $g = 0.0$ and $g = 1.0$, and evaluating the average consecutive delay objective function on all Monte Carlo samplings and calculating the difference in means.

Figure 5.10 shows the distribution of the average consecutive delay for different values of g for the operation changing the average objective the most. This operation corresponds to a dwell time as expected. Similar plots for the process time with the

[5]By evaluating estimates for o_i in increasing order, longest path calculations can be reduced to propagate changes on paths rooted in the o_i in the temporal networks.

greatest impact shows an almost undetectable shift in average objective values. This again confirms that while process times for trains travel summed up matters, their individual influence is easily drowned out by a single dwell time.

5.6 Conclusions

Robustness under disturbances is the ability to handle small stochastic variations in running and dwell times of trains in the network, without increasing output delays. In this chapter we applied the framework from chapter 4 to evaluating the robustness of a given solution to the train rescheduling problem. The robustness analysis can be carried out with a very small computational cost, thus it is suitable for the real-time evaluation of different alternative solutions. In this work the behaviour of three deterministic scheduling algorithms are evaluated in a stochastic environment.

The B&B solutions are nearly as robust to variability of input parameters as the myopic FIFO rule that relies on local information only. The greedy heuristic performs well when it delivers results, but must be deemed too unstable to use outside a framework such as B&B.

One aim of the chapter was to assess the influence of algorithms, stochaticity and sampling strategy, it turned out that the sampling strategy used to generate the deterministic instances solved, has the most significant effect on the solution quality. Strategies most closely resembling ignoring delays, were show to perform significantly worse than strategies corresponding to using expected values for each distribution. Furthermore it was show that slightly overestimating the expected value of the distributions is better to underestimation. This observation becomes important when the exact distributions are unknown.

B&B was originally feared to be more sensitive to stochasticity than FIFO due to it being a tighter schedule. We have shown however that B&B often retains and in some cases even improves its lead over FIFO after taking stochasticity into account.

Chapter 6

Conclusions

Contents

The initial concerns raised concerning real life scheduling problems were:

- Making companies aware of what science can provide, and at what cost.
- Cheaply and quickly assessing the magnitude of possible improvements to justify investments.
- Academic problems must be reformulated to take special constraints into consideration.
- Any stochasticity must be taken in account.
- Algorithms must be made ready for a production environment.

By working with companies directly, we have spread the word that we can deliver results, and further cooperation has been proposed by the companies. We have also significantly lowered the cost of assessing potential improvements by creating the framework presented in Chapter 4. This framework also allows us to address stochasticity using existing deterministic solvers. The academic problems must still be reformulated, but flexible models have been documented to work with the framework, easing the process. Finally, the reality module in the proposed framework makes deployment of the optimization algorithms smoother as it provides an interface that can be utilized to test the setup beforehand.

The rest of this chapter is a short summary of the work done, with emphasis on the possible future work.

6.1 Variable Size Bin Packing Problem

A specific implementation of a construction heuristic for the variable size bin packing problem was presented. The heuristic is currently being employed in a real life application where it has improved production line performance significantly by increasing the number of scales that could be utilized. Extensions of the construction heuristics were proposed to the anonymous company, but are not fully utilized in a production environment.

Furthermore a local search heuristic based on dynamic programming was presented. The heuristic is applicable to both problems with few item sizes and problems with many item sizes. It also benefits from an increased number of bin sizes, but have a respectable performance on classical bin packing problems. The solver was shown to consistently outperform state of the art methods under circumstances corresponding to the real life problem under consideration. The flexibility and expressiveness of the solver, the fast convergence and the "anytime" property makes the heuristic potentially adaptable for a range of real life variable size bin packing problems.

Danfoam's material usage has improved significantly after the proposed optimization methods were employed.

6.1.1 Future Work

A reactive approach based on storage scenarios was proposed as a research subject, but rejected by Danfoam due to the limited possible gain after the initial improvements. In cases with more precious materials this approach should be investigated further.

6.2 Scheduling Outages for Nuclear Power Plants

A solver for a large-scale real-life optimization problem has been presented. It uses a combination of constraint programming, greedy heuristics and local search heuristics.

The problem has multiple objectives that were weighted a priori and merged into a cost function. Due to the large scale of the problem, stochasticity is handled proactively.

The solutions obtained are competitive when compared to those found by other teams participating in the final evaluation of the ROADEF/EURO Challenge 2010. After fixing an implementation bug, our approach is robust in the sense that it is always able to find a feasible solution, and it achieves overall the best score, ranking first in the assessment procedure of the competition.

6.2.1 Future Work

As mentioned in the chapter, applying the reinsertion algorithm early in the search might speed up convergence of the algorithm and allow the local search to change

the number of outages. For this to be effective, a better estimate of the objective would be needed when reinserting outages. This could be based on the cost of type 1 production in the weeks considered for reinsertion.

Another possibility is to modulate when type 1 production is cheap in order to save fuel which can then be used in subsequent time steps where type 1 production is more expensive. It might also be possible to adjust the modulation phase such that a feasible minimum demand scenario is not necessary.

6.3 A Framework for Dynamic Rescheduling

The simulation based framework for dynamic rescheduling was presented and demonstrated on two problems: A Job Shop problem and a Single Machine Weighted Completion Time problem.

The framework is currently highly flexible and can be applied for:

- Evaluating robustness of solutions on multiple scenarios.
- Evaluate rescheduling strategies for stochastic scheduling problems.
- Implementing strategies outlined in Section 4.7.
- Facilitating decision makers by providing graphs for a decision support environment.
- Handling a real life rescheduling problem on site.

6.3.1 Future Work

Possible future applications are outlined in Section 4.7, but concrete plans exists to apply the framework to:

- Evaluate rescheduling strategies for the problem presented in Chapter 5.
- Applying the framework to the optimization of the real life anodizing plant described in [Skov 2007].

These applications are expected to result in a paper each.

Methods from parameter tuning such as the one presented in [Hutter 2009], could be adapted to the statistics gathering done with the framework, with the expectation of significantly fewer test runs before unfavourable configurations of the framework could be cut as statistically dominated.

Other possible work on the framework includes adapting it to manage resources by utilizing resource task networks. Utilizing simple confidence intervals to separate algorithms formally in case of closer performances. Finally, gradual methods of revealing the actual values behind the stochasticity could be implemented.

6.4 Robustness of Train Schedules

The framework was applied to a real life train scheduling problem presented in [Corman 2011b], was used to show that the B&B algorithm outperforms a myopic FIFO strategy after stochasticity was taken into consideration. Furthermore it was shown that the gap between the algorithms for average consecutive delays actually rose for some levels of stochasticity.

6.4.1 Future Work

Future research could be dedicated to the choice of sampling strategy. Furthermore the solution methods could be expanded into a closed loop rescheduling with emphasis on retaining a pool of candidate solutions to choose from and update, when stochastic data is revealed.

Appendix A

Solver Input and Output Files

A.1 Single Machine Weighted Completion Times Problem

A.1.1 Weights file

The first entry on each line is a unique identifier for the operation, the second entry is the associated weight.

```
J0 61
J1 21
J2 85
J3 87
J4 44
J5 35
J6 90
J7 65
J8 48
J9 87
J10 89
J11 90
J12 28
J13 14
J14 81
J15 30
J16 80
J17 69
J18 16
J19 21
J20 26
```

A.1.2 Instance and solution file

The instance and the solution files differ only by the ordering of the lines, and solution files are guaranteed to have the objective value set. The first entry on each line is a unique identifier of the operation, and the second is the associated duration.

```
OBJECTIVE: 542403
J3 11
J1 3
J2 23
J12 9
J14 52
J4 29
J19 14
J0 47
J9 72
J20 27
J8 57
J6 112
J10 112
J17 88
J11 126
J16 126
J7 119
J5 84
J15 99
J18 88
J13 101
```

A.1.3 Distribution file

The distribution file contains a textual representation of the distribution of each operations duration.

```
<Uniform42:63;J0>
<Uniform3:5;J1>
<Uniform20:30;J2>
<Uniform8:12;J3>
<Uniform24:36;J4>
<Uniform62:93;J5>
<Uniform93:140;J6>
<Uniform99:149;J7>
<Uniform43:65;J8>
<Uniform60:90;J9>
<Uniform83:125;J10>
<Uniform84:126;J11>
<Uniform8:12;J12>
<Uniform90:135;J13>
<Uniform42:63;J14>
<Uniform83:125;J15>
<Uniform89:134;J16>
<Uniform75:113;J17>
<Uniform88:132;J18>
<Uniform10:15;J19>
<Uniform22:33;J20>
```

A.2 Job Shop Problem

A.2.1 Model file

The model file is based on the constraint programming job shop model included in the ILOG CP optimizer. Release times and distribution identifiers have been added.

```
using CP;

int nbJobs = ...;
int nbMchs = ...;

range Jobs = 0..nbJobs-1;
range Mchs = 0..nbMchs-1;
// Mchs is used both to index machines and operation position in job

tuple Operation {
  int mch; // Machine
  int pt;  // Processing time
  string ptdist; // Distribution for the processing time
  int rd;  // Release date
};

Operation Ops[j in Jobs][m in Mchs] = ...;

dvar interval itvs[j in Jobs][o in Mchs] size Ops[j][o].pt;
dvar sequence mchs[m in Mchs] in all(j in Jobs, o in Mchs : Ops[j][o].mch == m) itvs[j][o];

execute {
  cp.param.TimeLimit = 60;
  cp.param.Workers = 1;
  cp.param.LogPeriod = 100000;
  //cp.param.LogVerbosity = "Quiet";
}

minimize max(j in Jobs) endOf(itvs[j][nbMchs-1]);
subject to {
  forall (m in Mchs)
    noOverlap(mchs[m]);
  forall (j in Jobs, o in 0..nbMchs-2)
    endBeforeStart(itvs[j][o], itvs[j][o+1]);
  forall (j in Jobs, o in 0..nbMchs-1)
    startOf(itvs[j][o])>=Ops[j][o].rd;
}

execute {
  for (var j = 0; j <= nbJobs-1; j++) {
    for (var o = 0; o <= nbMchs-1; o++) {
      write(itvs[j][o].start + " ");
    }
    writeln("");
  }
}
```

A.2.2 Instance file

The data is provided as an array of arrays of operations. Each operation has an associated machine, a duration, an identifier and a release time respectively.

```
nbJobs = 10;
nbMchs = 10;

Ops = [
[ <0,29,j0o0,0>, <1,78,j0o1,0>, <2,9,j0o2,0>, <3,36,j0o3,0>, <4,49,j0o4,0>,
 <5,11,j0o5,0>, <6,62,j0o6,0>, <7,56,j0o7,0>, <8,44,j0o8,0>, <9,21,j0o9,0> ],
[ <0,43,j1o0,0>, <2,90,j1o1,0>, <4,75,j1o2,0>, <9,11,j1o3,0>, <3,69,j1o4,0>,
<1,28,j1o5,0>, <6,46,j1o6,0>, <5,46,j1o7,0>, <7,72,j1o8,0>, <8,30,j1o9,0> ],
[ <1,91,j2o0,0>, <0,85,j2o1,0>, <3,39,j2o2,0>, <2,74,j2o3,0>, <8,90,j2o4,0>,
<5,10,j2o5,0>, <7,12,j2o6,0>, <6,89,j2o7,0>, <9,45,j2o8,0>, <4,33,j2o9,0> ],
[ <1,81,j3o0,0>, <2,95,j3o1,0>, <0,71,j3o2,0>, <4,99,j3o3,0>, <6,9,j3o4,0>,
<8,52,j3o5,0>, <7,85,j3o6,0>, <3,98,j3o7,0>, <9,22,j3o8,0>, <5,43,j3o9,0> ],
[ <2,14,j4o0,0>, <0,6,j4o1,0>, <1,22,j4o2,0>, <5,61,j4o3,0>, <3,26,j4o4,0>,
<4,69,j4o5,0>, <8,21,j4o6,0>, <7,49,j4o7,0>, <9,72,j4o8,0>, <6,53,j4o9,0> ],
[ <2,84,j5o0,0>, <1,2,j5o1,0>, <5,52,j5o2,0>, <3,95,j5o3,0>, <8,48,j5o4,0>,
<9,72,j5o5,0>, <0,47,j5o6,0>, <6,65,j5o7,0>, <4,6,j5o8,0>, <7,25,j5o9,0> ],
[ <1,46,j6o0,0>, <0,37,j6o1,0>, <3,61,j6o2,0>, <2,13,j6o3,0>, <6,32,j6o4,0>,
<5,21,j6o5,0>, <9,32,j6o6,0>, <8,89,j6o7,0>, <7,30,j6o8,0>, <4,55,j6o9,0> ],
[ <2,31,j7o0,0>, <0,86,j7o1,0>, <1,46,j7o2,0>, <5,74,j7o3,0>, <4,32,j7o4,0>,
<6,88,j7o5,0>, <8,19,j7o6,0>, <9,48,j7o7,0>, <7,36,j7o8,0>, <3,79,j7o9,0> ],
[ <0,76,j8o0,0>, <1,69,j8o1,0>, <3,76,j8o2,0>, <5,51,j8o3,0>, <2,85,j8o4,0>,
<9,11,j8o5,0>, <6,40,j8o6,0>, <7,89,j8o7,0>, <4,26,j8o8,0>, <8,74,j8o9,0> ],
[ <1,85,j9o0,0>, <0,13,j9o1,0>, <2,61,j9o2,0>, <6,7,j9o3,0>, <8,64,j9o4,0>,
<9,76,j9o5,0>, <5,47,j9o6,0>, <3,52,j9o7,0>, <4,90,j9o8,0>, <7,45,j9o9,0> ]
];
```

A.2.3 Solution file

The solution file containing the start times of each operation. Omitted lines have been marked with "...".

```
IBM ILOG CPLEX Optimization Studio Academic Research Edition

<<< setup


<<< generate

 ! ------------------------------------------------------------------------
 ! Minimization problem - 110 variables, 200 constraints
 ! Preprocessing : 100 extractables eliminated
 ! LogPeriod            = 100.000
 ! Workers              = 2
 ! TimeLimit            = 300
 ! Initial process time : 0,00s (0,00s extraction + 0,00s propagation)
 !  . Log search space  : 664,4 (before), 664,4 (after)
 !  . Memory usage      : 690,8 Kb (before), 925,5 Kb (after)
 ! ------------------------------------------------------------------------
...
 ! ------------------------------------------------------------------------
 ! Solution status        : Terminated by limit, solution found
 ! Number of branches     : 9.943.919
 ! Number of fails        : 4.087.288
 ! Total memory usage     : 6,6 Mb (4,2 Mb CP Optimizer + 2,4 Mb Concert)
 ! Time spent in solve    : 300,01s (300,01s engine + 0,00s extraction)
 ! Search speed (br. / s) : 33.145,3
 ! ------------------------------------------------------------------------
 ! Search terminated, replaying best solution found

<<< solve


OBJECTIVE: 937
342 442 526 566 602 651 662 755 811 904
162 436 526 601 612 681 724 770 816 888
351 442 527 566 640 730 743 770 859 904
0 115 210 281 380 389 518 733 831 853
210 281 329 351 412 438 507 603 684 859
31 150 152 226 321 373 527 574 651 682
152 287 324 423 436 468 489 528 652 683
0 76 198 277 380 468 617 636 707 831
0 81 150 226 277 362 389 429 657 730
244 329 362 429 441 521 597 681 738 888

<<< post process


<<< done
```

A.2.4 Distribution file

```
<Uniform29:44;j0o0>
<Uniform78:117;j0o1>
<Uniform9:14;j0o2>
<Uniform36:54;j0o3>
<Uniform49:74;j0o4>
<Uniform11:17;j0o5>
<Uniform62:93;j0o6>
<Uniform56:84;j0o7>
<Uniform44:66;j0o8>
<Uniform21:32;j0o9>
<Uniform43:65;j1o0>
<Uniform90:135;j1o1>
<Uniform75:113;j1o2>
<Uniform11:17;j1o3>
<Uniform69:104;j1o4>
<Uniform28:42;j1o5>
<Uniform46:69;j1o6>
<Uniform46:69;j1o7>
<Uniform72:108;j1o8>
<Uniform30:45;j1o9>
<Uniform91:137;j2o0>
<Uniform85:128;j2o1>
<Uniform39:59;j2o2>
<Uniform74:111;j2o3>
<Uniform90:135;j2o4>
<Uniform10:15;j2o5>
<Uniform12:18;j2o6>
<Uniform89:134;j2o7>
<Uniform45:68;j2o8>
<Uniform33:50;j2o9>
<Uniform81:122;j3o0>
<Uniform95:143;j3o1>
<Uniform71:107;j3o2>
<Uniform99:149;j3o3>
<Uniform9:14;j3o4>
<Uniform52:78;j3o5>
<Uniform85:128;j3o6>
<Uniform98:147;j3o7>
<Uniform22:33;j3o8>
<Uniform43:65;j3o9>
<Uniform14:21;j4o0>
<Uniform6:9;j4o1>
<Uniform22:33;j4o2>
<Uniform61:92;j4o3>
<Uniform26:39;j4o4>
<Uniform69:104;j4o5>
<Uniform21:32;j4o6>
<Uniform49:74;j4o7>
<Uniform72:108;j4o8>
<Uniform53:80;j4o9>
<Uniform84:126;j5o0>
<Uniform2:3;j5o1>
<Uniform52:78;j5o2>
<Uniform95:143;j5o3>
<Uniform48:72;j5o4>
<Uniform72:108;j5o5>
<Uniform47:71;j5o6>
<Uniform65:98;j5o7>
<Uniform6:9;j5o8>
<Uniform25:38;j5o9>
```

```
<Uniform46:69;j6o0>
<Uniform37:56;j6o1>
<Uniform61:92;j6o2>
<Uniform13:20;j6o3>
<Uniform32:48;j6o4>
<Uniform21:32;j6o5>
<Uniform32:48;j6o6>
<Uniform89:134;j6o7>
<Uniform30:45;j6o8>
<Uniform55:83;j6o9>
<Uniform31:47;j7o0>
<Uniform86:129;j7o1>
<Uniform46:69;j7o2>
<Uniform74:111;j7o3>
<Uniform32:48;j7o4>
<Uniform88:132;j7o5>
<Uniform19:29;j7o6>
<Uniform48:72;j7o7>
<Uniform36:54;j7o8>
<Uniform79:119;j7o9>
<Uniform76:114;j8o0>
<Uniform69:104;j8o1>
<Uniform76:114;j8o2>
<Uniform51:77;j8o3>
<Uniform85:128;j8o4>
<Uniform11:17;j8o5>
<Uniform40:60;j8o6>
<Uniform89:134;j8o7>
<Uniform26:39;j8o8>
<Uniform74:111;j8o9>
<Uniform85:128;j9o0>
<Uniform13:20;j9o1>
<Uniform61:92;j9o2>
<Uniform7:11;j9o3>
<Uniform64:96;j9o4>
<Uniform76:114;j9o5>
<Uniform47:71;j9o6>
<Uniform52:78;j9o7>
<Uniform90:135;j9o8>
<Uniform45:68;j9o9>
```

A.3 Train Scheduling Problem

A.3.1 Instance file

Cut out parts of the instance marked.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"  href="html.xsl"?>
<!DOCTYPE root [
    <!ELEMENT root (XMLversion, instance*, resource+, path*, time_point*, time_profile*,
routed_job*, job*, lot*, setup_removal_table*, machine*, connection*)>
    <!ELEMENT XMLversion (#PCDATA)>
    <!ELEMENT instance (#PCDATA)>
    <!ELEMENT resource (id_resource, name*, resource_type, info1*, info2*)>
    <!ELEMENT id_resource (#PCDATA)>
    <!ELEMENT resource_type (#PCDATA)>
    <!ELEMENT info1 (#PCDATA)>
    <!ELEMENT info2 (#PCDATA)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT path (id_path, resource_list_elem+)>
    <!ELEMENT resource_list_elem (id_resource)>
    <!ELEMENT id_path (#PCDATA)>
    <!ELEMENT time_point (id_time_point, processing_time*, perishability_time*,release_time*,
due_date_time*, deadline_time*, setup_time*, removal_time*, dependent_setup_type*,
dependent_removal_type*)>
    <!ELEMENT id_time_point (#PCDATA)>
    <!ELEMENT processing_time (#PCDATA)>
    <!ELEMENT perishability_time (#PCDATA)>
    <!ELEMENT release_time (#PCDATA)>
    <!ELEMENT due_date_time (#PCDATA)>
    <!ELEMENT deadline_time (#PCDATA)>
    <!ELEMENT setup_time (#PCDATA)>
    <!ELEMENT removal_time (#PCDATA)>
    <!ELEMENT dependent_setup_type (#PCDATA)>
    <!ELEMENT dependent_removal_type (#PCDATA)>
    <!ELEMENT time_profile (id_time_profile, id_path, time_point_list_elem+)>
    <!ELEMENT time_point_list_elem (id_time_point)>
    <!ELEMENT id_time_profile (#PCDATA)>
    <!ELEMENT job (id_job, name*, routed_job_list_elem+)>
    <!ELEMENT routed_job (id_routed_job, job_type, id_time_profile)>
    <!ELEMENT routed_job_list_elem (id_routed_job)>
    <!ELEMENT id_job (#PCDATA)>
    <!ELEMENT id_routed_job (#PCDATA)>
    <!ELEMENT job_type (#PCDATA)>
    <!ELEMENT lot (id_lot, id_routed_job, lot_dimension, id_resource_nowait*)>
    <!ELEMENT id_lot (#PCDATA)>
    <!ELEMENT lot_dimension (#PCDATA)>
    <!ELEMENT id_resource_nowait (#PCDATA)>
    <!ELEMENT setup_removal_table (table_entry+)>
    <!ELEMENT table_entry (dependent_type1, dependent_type2,  setup_removal_type,
dependent_time)>
    <!ELEMENT dependent_type1 (#PCDATA)>
    <!ELEMENT dependent_type2 (#PCDATA)>
    <!ELEMENT setup_removal_type (#PCDATA)>
    <!ELEMENT dependent_time (#PCDATA)>
    <!ELEMENT machine (resource_list_elem+)>
    <!ELEMENT connection (id_timepoint_from, id_timepoint_to, connection_weight,
connection_price, connection_type)>
    <!ELEMENT id_timepoint_from (#PCDATA)>
    <!ELEMENT id_timepoint_to (#PCDATA)>
    <!ELEMENT connection_weight (#PCDATA)>
    <!ELEMENT connection_price (#PCDATA)>
```

```
    <!ELEMENT connection_type (#PCDATA)>
]>
<root>
<XMLversion>Adamo-2011-03-22</XMLversion>
      <resource>
              <id_resource>1</id_resource>
                 <name>1621T@Ut-1621T@Ut</name>
                 <resource_type>blocking</resource_type>
      </resource>
      <resource>
              <id_resource>2</id_resource>
                 <name>1633AT@Utge-1634CT@Utva</name>
                 <resource_type>blocking</resource_type>
      </resource>
//Cut resources
      <resource>
              <id_resource>504</id_resource>
                 <name>end path 183</name>
                 <resource_type>blocking</resource_type>
      </resource>

    <path>
          <id_path>1</id_path>
          <resource_list_elem>
           <id_resource>1</id_resource>
          </resource_list_elem>
          <resource_list_elem>
           <id_resource>322</id_resource>
          </resource_list_elem>
          <resource_list_elem>
           <id_resource>2</id_resource>
          </resource_list_elem>
          <resource_list_elem>
           <id_resource>3</id_resource>
          </resource_list_elem>
//cut resource_list_elems
          <resource_list_elem>
           <id_resource>422</id_resource>
          </resource_list_elem>
    </path>
    <path>
//cut paths

    <time_point>
<id_time_point>1001001</id_time_point>
<processing_time>0</processing_time>
<release_time>3300</release_time>
<removal_time>46</removal_time>
   </time_point>
   <time_point>
<id_time_point>1001002</id_time_point>
<processing_time>120</processing_time>
        <due_date_time>3300</due_date_time>
        <removal_time>46</removal_time>
   </time_point>
//cut time_points

   <time_profile>
    <id_time_profile>1</id_time_profile>
        <id_path>1</id_path>
        <time_point_list_elem>
         <id_time_point>1001001</id_time_point>
```

```
</time_point_list_elem>
<time_point_list_elem>
<id_time_point>1001002</id_time_point>
        </time_point_list_elem>
//cut time_point_list_elems
    </time_profile>
// cut time prifiles

    <routed_job>
            <id_routed_job>1</id_routed_job>
            <job_type>normal_job</job_type>
            <id_time_profile>1</id_time_profile>
    </routed_job>
// cut routed jobs

    <job>
            <id_job>1</id_job>
            <name>0B60001</name>
            <routed_job_list_elem>
                <id_routed_job>1</id_routed_job>
            </routed_job_list_elem>
    </job>
// cut jobs

      <machine>
          <resource_list_elem>
                <id_resource>1</id_resource>
          </resource_list_elem>
      </machine>
// cut macines
      <machine>
          <resource_list_elem>
                <id_resource>96</id_resource>
          </resource_list_elem>
          <resource_list_elem>
                <id_resource>187</id_resource>
          </resource_list_elem>
          <resource_list_elem>
                <id_resource>223</id_resource>
          </resource_list_elem>
          <resource_list_elem>
                <id_resource>236</id_resource>
          </resource_list_elem>
          <resource_list_elem>
                <id_resource>239</id_resource>
          </resource_list_elem>
          <resource_list_elem>
                <id_resource>281</id_resource>
          </resource_list_elem>
      </machine>
// cut machines
</root>
```

A.3.2 Solution file

Cut out parts of the solution marked.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet version="1.0" type="text/xsl" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 href="html.xsl" ?>
<!DOCTYPE root [
<!ELEMENT root (machine, job, problem, operation+*, arc+*)>
<!ELEMENT machine (machine_elem+)>
<!ELEMENT machine_elem (id_resource, name*, from*, to*)>
<!ELEMENT id_resource (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT job (job_elem+)>
<!ELEMENT job_elem (id_job, name*, color*, direction*)>
<!ELEMENT id_job (#PCDATA)>
<!ELEMENT problem (instance_name, size_node, size_arc, size_pair, makespan)>
<!ELEMENT instance_name (#PCDATA)>
<!ELEMENT size_node (#PCDATA)>
<!ELEMENT size_arc (#PCDATA)>
<!ELEMENT size_pair (#PCDATA)>
<!ELEMENT makespan (#PCDATA)>
<!ELEMENT operation (id_operation, id_job, id_resource, head, proc, tail, finish, due*,
release*)>
<!ELEMENT id_operation (#PCDATA)>
<!ELEMENT head (#PCDATA)>
<!ELEMENT proc (#PCDATA)>
<!ELEMENT tail (#PCDATA)>
<!ELEMENT finish (#PCDATA)>
<!ELEMENT due (#PCDATA)>
<!ELEMENT release (#PCDATA)>
<!ELEMENT arc (id_arc, weight, id_prev_op, id_next_op, selected, pair)>
<!ELEMENT id_arc (#PCDATA)>
<!ELEMENT weight (#PCDATA)>
<!ELEMENT id_prev_op (#PCDATA)>
<!ELEMENT id_next_op (#PCDATA)>
<!ELEMENT selected (#PCDATA)>
<!ELEMENT pair (#PCDATA)>
]>
<root>
<machine>
    <machine_elem>
        <id_resource>504</id_resource>
        <name>end path 183</name>
    </machine_elem>
    <machine_elem>
        <id_resource>503</id_resource>
        <name>end path 182</name>
    </machine_elem>
//cut machines_elems
</machine>
<job>
    <job_elem>
        <id_job>82</id_job>
        <name>0RBH40S</name>
    </job_elem>
    <job_elem>
        <id_job>81</id_job>
        <name>0OVF11</name>
    </job_elem>
// cut job_elems
</job>
<problem>
```

```
    <instance_name>ht.xml.3600.routes.xml</instance_name>
    <size_node>2585</size_node>
    <size_arc>16221</size_arc>
    <size_pair>6620</size_pair>
    <makespan>0</makespan>
</problem>
<operation>
    <id_operation>1001037</id_operation>
    <id_job>1</id_job>
    <id_resource>422</id_resource>
    <head>4907</head>
    <proc>0</proc>
    <tail>-4920</tail>
    <finish>0</finish>
    <due>4920</due>
</operation>
// cut operations
<operation>
    <id_operation>1009044</id_operation>
    <id_job>9</id_job>
    <id_resource>67</id_resource>
    <head>4241</head>
    <proc>44</proc>
    <tail>-4351</tail>
    <finish>44</finish>
</operation>
// cut operations
<arc>
    <id_arc>2980</id_arc>
    <weight>0</weight>
    <id_prev_op>1001037</id_prev_op>
    <id_next_op>-2583</id_next_op>
    <selected>1</selected>
    <pair>-1</pair>
</arc>
// cut arcs
<arc>
    <id_arc>3561</id_arc>
    <weight>46</weight>
    <id_prev_op>1054004</id_prev_op>
    <id_next_op>1078033</id_next_op>
    <selected>0</selected>
    <pair>291</pair>
</arc>
<arc>
    <id_arc>3562</id_arc>
    <weight>46</weight>
    <id_prev_op>1078035</id_prev_op>
    <id_next_op>1054002</id_next_op>
    <selected>1</selected>
    <pair>291</pair>
</arc>
// cut arcs
</root>
```

A.3.3 Distribution file

Some distributions were cut out of the file to conserve space.

```
<Singular0;1017015>
<Uniform25:27;1017014>
<Uniform18:19;1017013>
<Singular0;1017012>
<Uniform31:34;1017011>
<Weibull;261;2.0824;4;1017010>
<Uniform53:57;1017018>
<Singular0;1043012>
<Uniform24:26;1017019>
<Weibull;261;2.0824;4;1043011>
<Uniform31:34;1017016>
<Uniform20:22;1043010>
<Singular0;1017017>
<Singular0;1017002>
<Singular3;1017001>
<Singular6;1017004>
<Uniform28:30;1017003>
<Weibull;261;2.0824;4;1034008>
<Singular0;1043007>
<Singular0;1034009>
<Singular7;1043006>
<Uniform21:23;1043009>
<Uniform28:30;1043008>
<Uniform36:39;1017009>
<Singular0;1034004>
<Uniform34:37;1043003>
<Uniform21:23;1034005>
<Weibull;261;2.0824;4;1043002>
<Uniform33:36;1034006>
<Uniform29:32;1043005>
<Uniform48:51;1034007>
<Uniform29:32;1043004>
<Singular0;1017005>
<Uniform16:17;1017006>
<Singular0;1034001>
<Uniform59:63;1017007>
<Weibull;261;2.0824;4;1034002>
// cut remaining distributions
```

Appendix B

Stochastic and Deterministic Gantt Charts

This appendix documents select outputs from a run of the framework for dynamic rescheduling on the Job Shop instance FT10 modified with an $extend = 0.8$. The solver uses the sampling strategy $g = 0.5$, and is allowed to reschedule every 10 seconds ($f = 10$), but will only do so if the objective function has changed $T_{Objective}$. It obtains knowledge about an operations duration as it starts $k = 0$.

Each figure documents a the state of the system in a specific time step t. The deterministic Gantt chart (bottom) is the solution the solver produced, modified for any changed durations. The stochastic Gantt chart (top) is the scenarios applied to the same solution.[1] The vertical line denotes the current time t.

As time progresses, the operations represented in the upper plot are assigned deterministic process times one after one, and the added information propagates through the plot making the subsequent operations start (and consequently also end-) times less uncertain.

[1] Not that this differs from the plots in Chapter 4 where the two plots represents two different time steps.

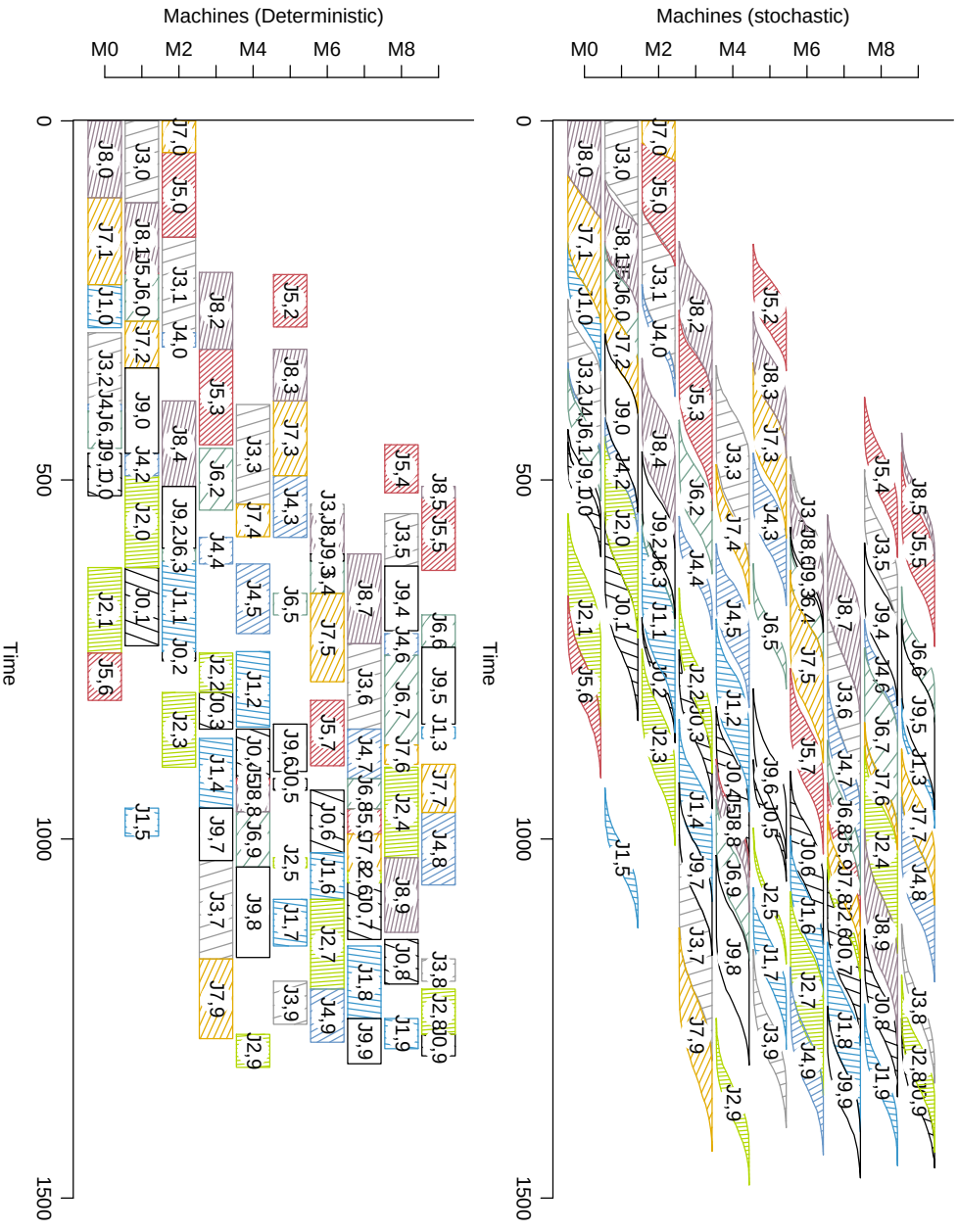

Figure B.1: The state of the system at time $t = -1$. No durations of jobs have been revealed.

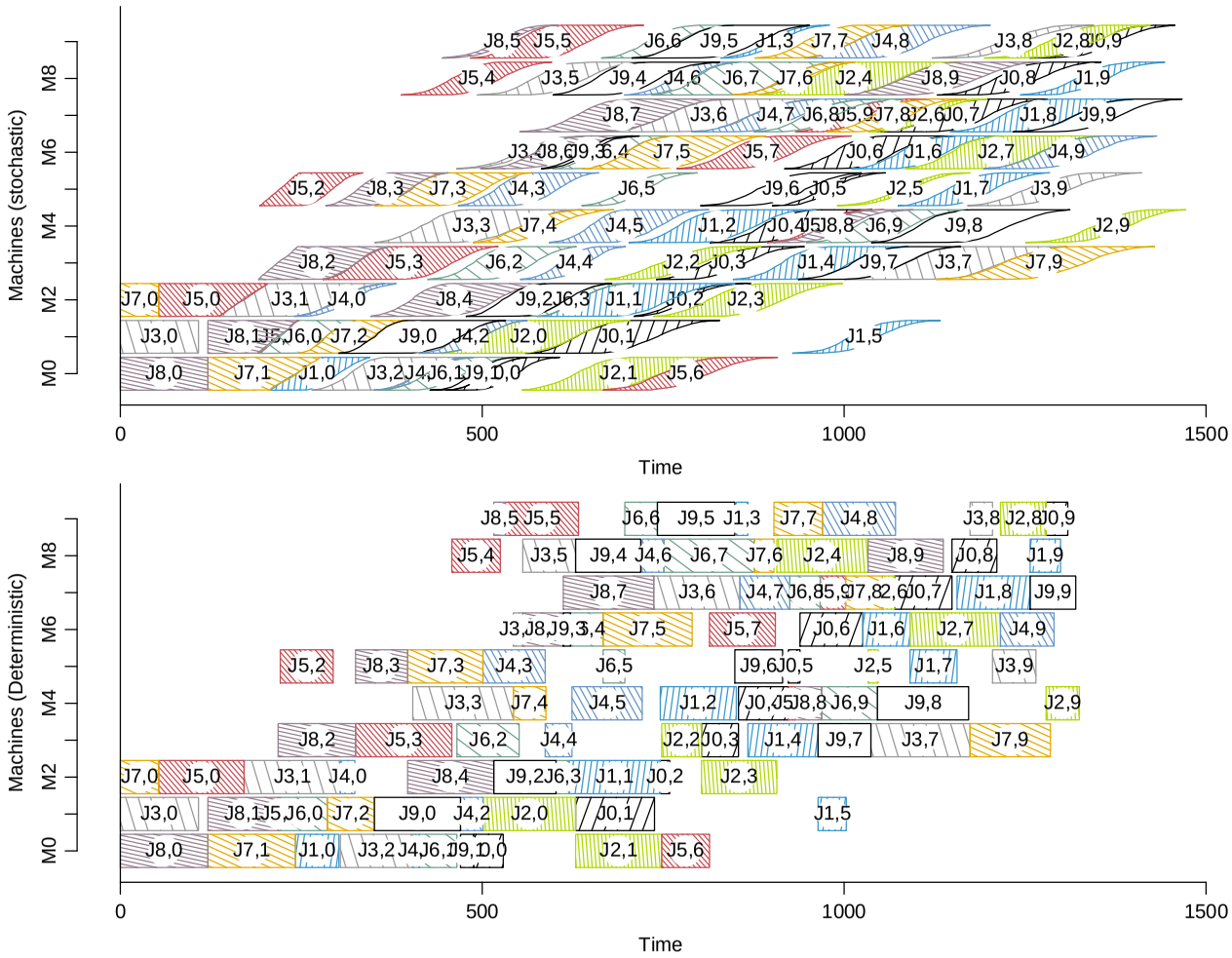

Figure B.2: The state of the system at time $t = 0$. The durations of the operations represented by the shapes $J7, 0$, $J3, 0$ and $J8, 0$ has just been revealed to the framework. Not that this also uniquely determines $J8, 1$'s start time, while $J3, 1$ still has multiple possible start times.

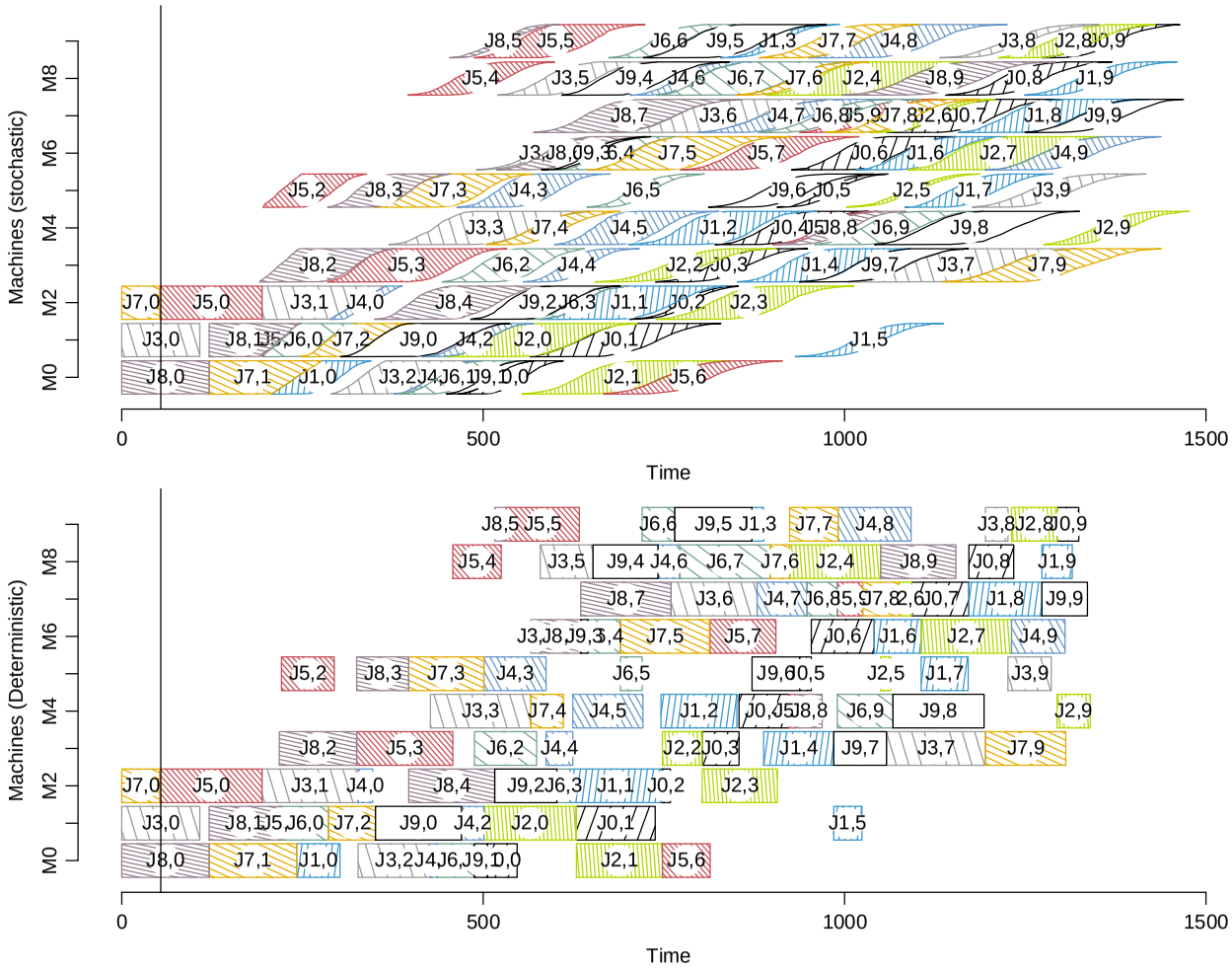

Figure B.3: Next rescheduling occurs at $t = 54$ as the first change since $t = 0$ occurs then: $J5, 0$'s duration becomes known. This rescheduling did not change the solution.

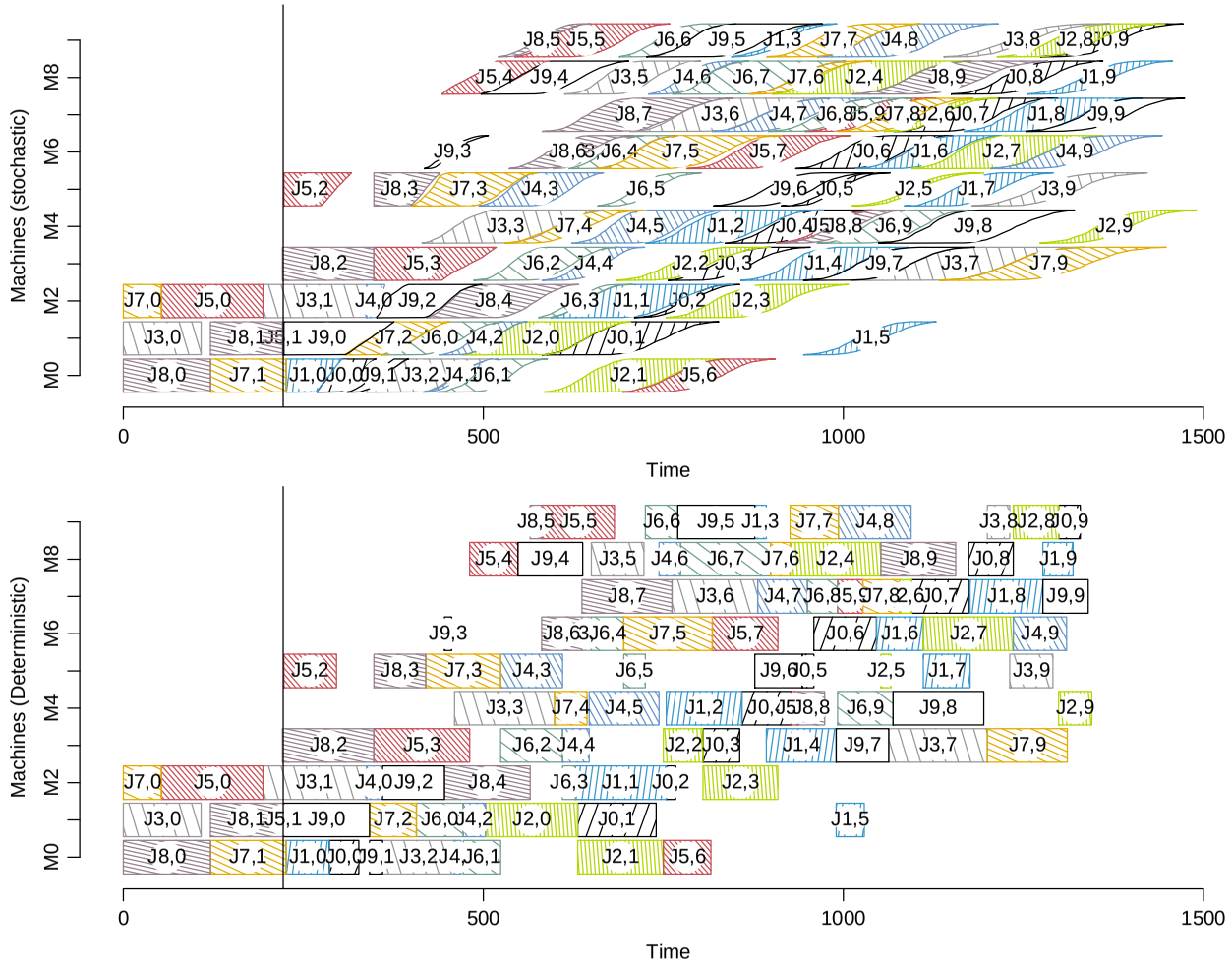

Figure B.4: Fast forwarding to $t = 222$ past other reschedulings, we detect the first rescheduling that changed the solution ($J9, 3$ was moved).

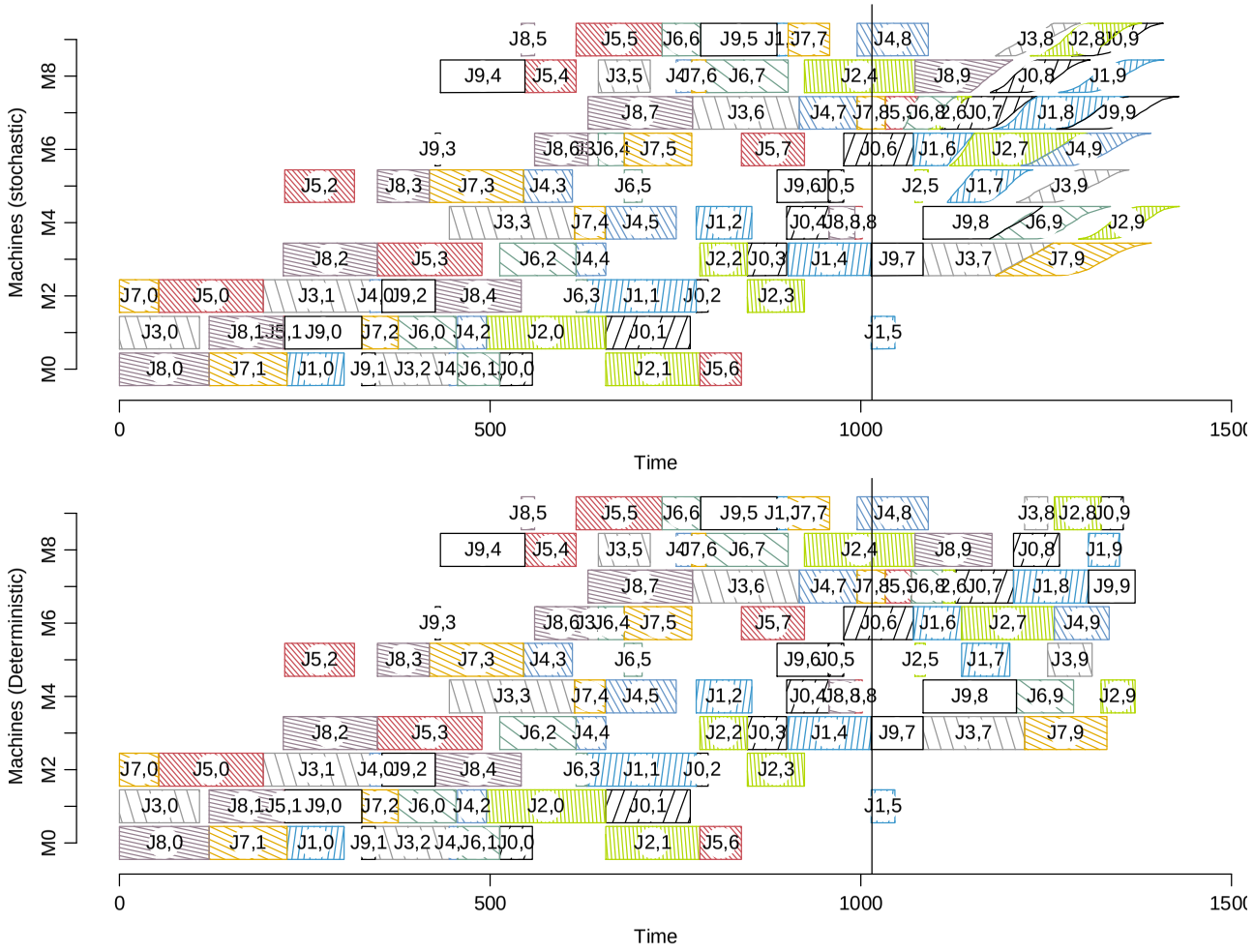

Figure B.5: At time $t = 1015$ we see that the stochasticity has dropped significantly, and that the ordering on $M8$ has changed since $t = 222$ (see $J7, 6$). Not that the part of the two Gantt charts left of the time line are identical.

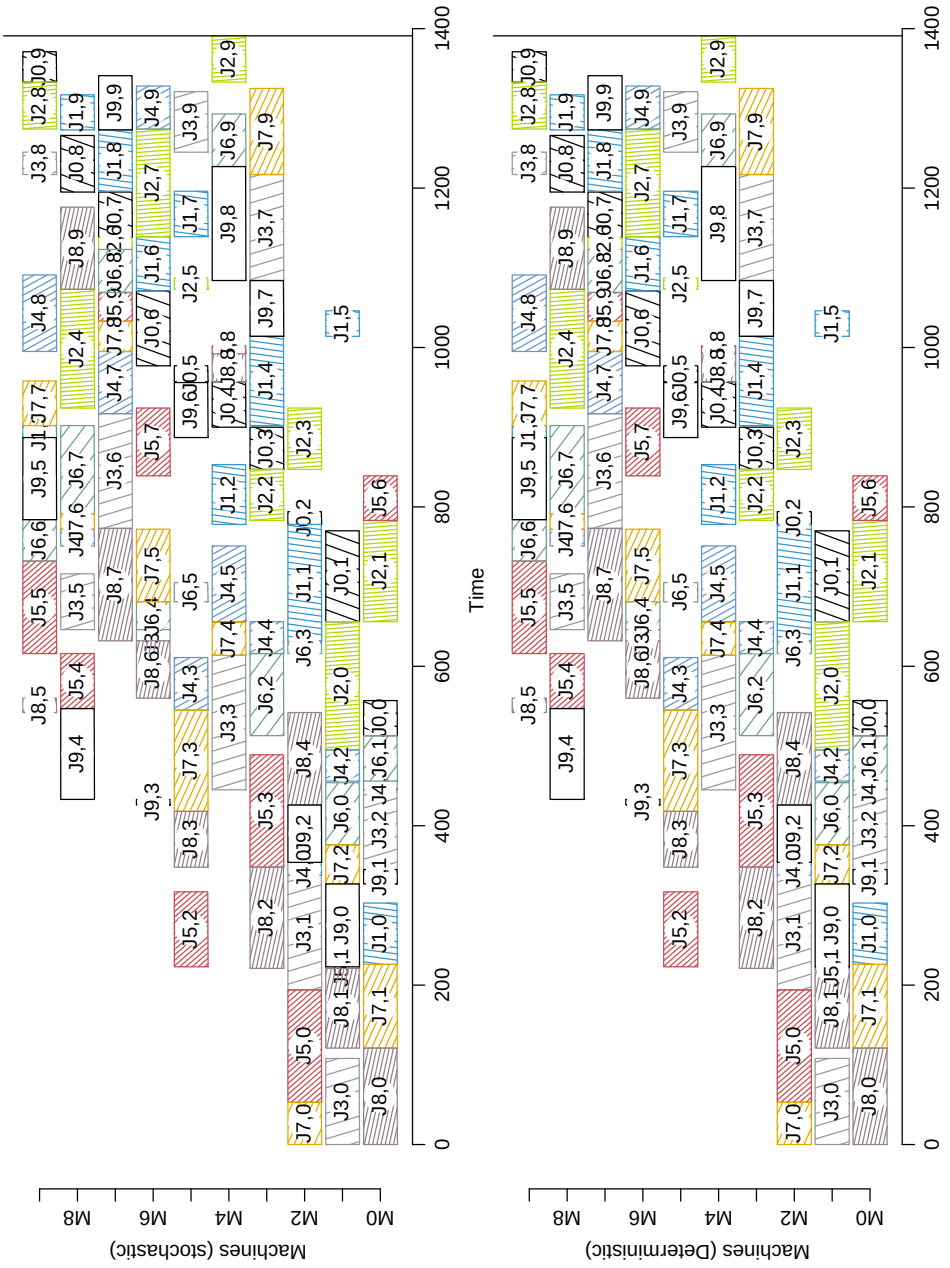

Figure B.6: At time $t = 1391$ the scheduled has been executed, and simulation stops.

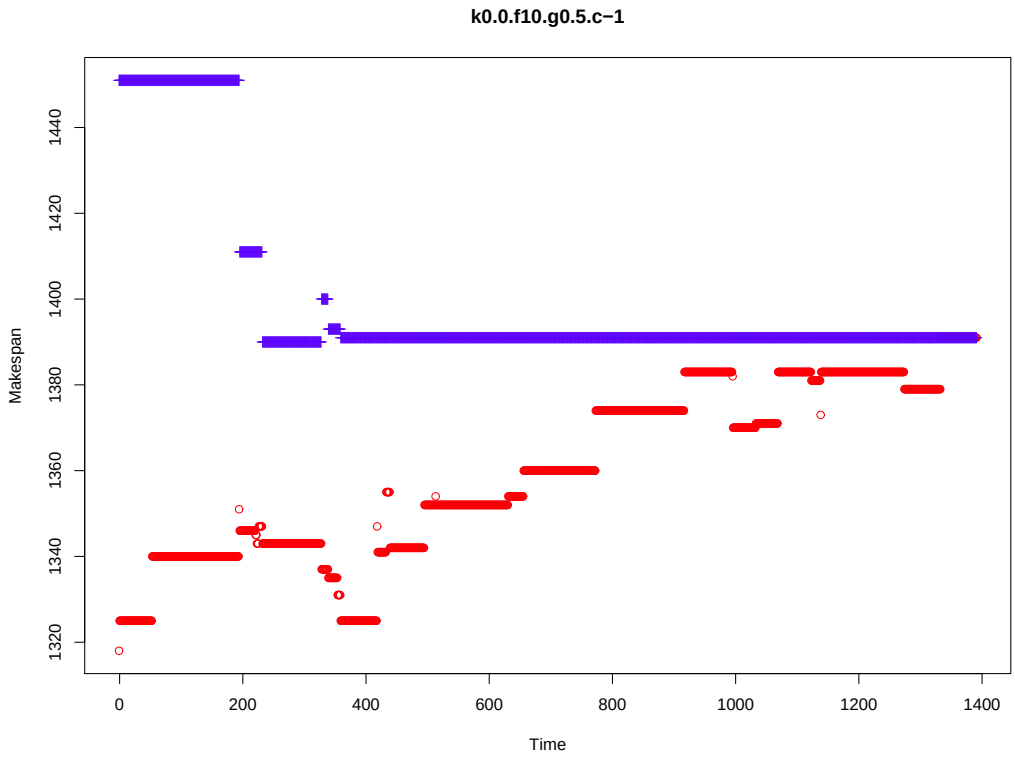

Figure B.7: During the execution of the framework the objective changed as shown. The red marks represents the perceived deterministic makespan, while the blue marks the objective that would have been obtained by not rescheduling from that point in time.

Appendix C

Train Robustness

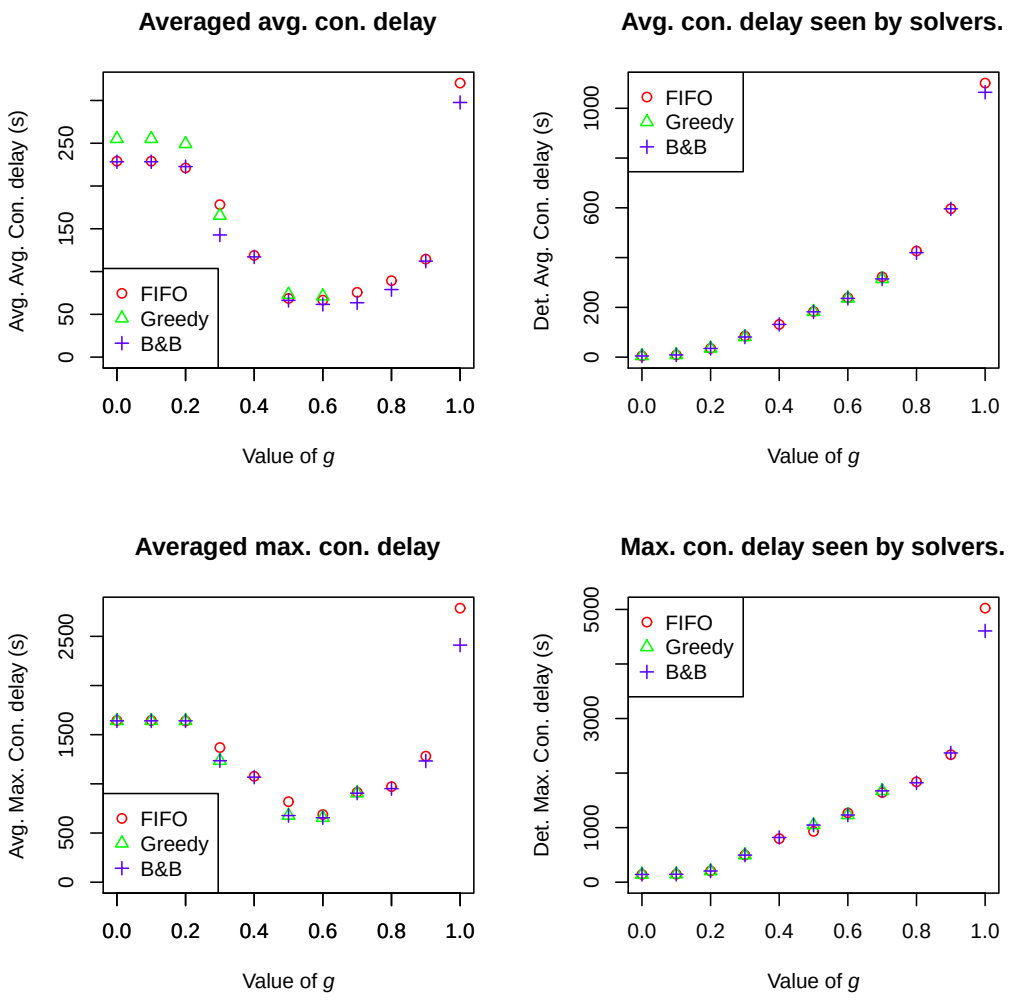

Figure C.1: Average values of consecutive delays as a function of the sampling strategy (g) for the off peak scenario. The two plots to the right is the solution quality produced by the solvers, and the two plots to the left is the quality after the Monte Carlo samplings are applied to the same solutions.

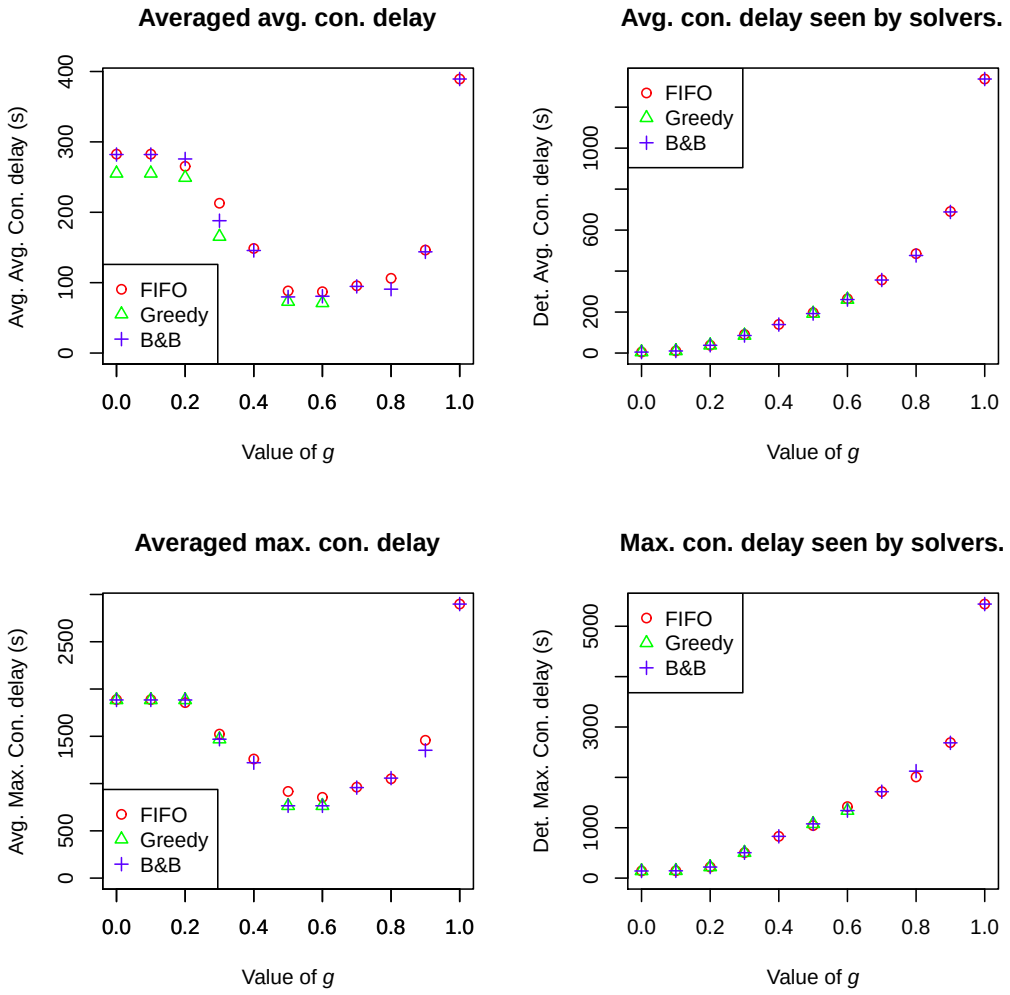

Figure C.2: Average values of consecutive delays as a function of the sampling strategy (g) for the on peak scenario. The two plots to the right is the solution quality produced by the solvers, and the two plots to the left is the quality after the Monte Carlo samplings are applied to the same solutions.

		Deterministic			Stochastic		
Stoch.	Obj. func.	FIFO	B&B	%	FIFO	B&B	%
$[-2.5\%, 5\%]$	Avg. Late.	235.03	231.86	1.35	261.55	256.51	1.93
	Max. Late.	1199.00	1199.00	0.00	1548.34	1550.16	-0.12
	Avg. Con.	30.13	26.95	10.53	73.24	68.21	6.88
	Max. Con.	291	275	5.50	722.44	706.37	2.22
$[-5\%, 10\%]$	Avg. Late.	245.71	242.90	1.15	268.12	263.18	1.84
	Max. Late.	1330.50	1247	6.28	1597.63	1573.42	1.52
	Avg. Con.	31.90	29.08	8.83	74.81	69.88	6.59
	Max. Con.	328.50	302.00	8.07	743.55	703.84	5.34
$[-7.5\%, 15\%]$	Avg. Late.	254.66	253.16	0.59	275.07	270.48	1.67
	Max. Late.	1364.75	1291.00	5.40	1621.66	1596.72	1.54
	Avg. Con.	33.29	31.79	4.52	76.94	72.36	5.96
	Max. Con.	333.75	325.75	2.40	746.8	706.36	5.42
$[-10\%, 20\%]$	Avg. Late.	271.54	266.23	1.96	285.94	277.70	2.88
	Max. Late.	1481.75	1405.50	5.15	1679.11	1642.19	2.20
	Avg. Con.	39.41	34.10	13.47	82.47	74.24	9.99
	Max. Con.	542.00	335.25	38.15	871.37	724.57	16.85

Table C.1: Average results on deterministic and stochastic instances for $g = 0.6$, averaged over off/on peak cases.

Bibliography

[Alves 2008] Cláudio Alves and J.M. Valério de Carvalho. *A stabilized branch-and-price-and-cut algorithm for the multiple length cutting stock problem.* Comput. Oper. Res.Comput. Oper. Res., vol. 35, no. 4, pages 1315 – 1328, 2008. (Cited on pages 15 and 19.)

[Applegate 2002] David L. Applegate, Luciana S. Buriol, Bernard L. Dillard, David S. Johnson and Peter W. Shor. *The Cutting-Stock Approach to Bin Packing: Theory and Experiments.* In Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments, pages 1–25, 2002. (Cited on page 26.)

[Apt 2003] Krzysztof R. Apt. Principles of constraint programming. Cambridge university press, 2003. (Cited on page 49.)

[Aytug 2005] Haldun Aytug, Mark A. Lawley, Kenneth McKay, Shantha Mohan and Reha Uzsoy. *Executing production schedules in the face of uncertainties: A review and some future directions.* European Journal of Operational Research, vol. 161, no. 1, pages 86 – 110, 2005. (Cited on page 79.)

[Baker 2009] Kenneth R. Baker and Dan Trietsch. Principles of sequencing and scheduling. Wiley Publishing, 2009. (Cited on page 84.)

[Balas 1979] E. Balas. *Disjunctive programming.* Annals of Discrete Mathematics, vol. 5, pages 3 – 51, 1979. (Cited on page 77.)

[Bang-Jensen 2012] J Bang-Jensen and R Larsen. *Efficient algorithms for real-life instances of the variable size bin packing problem.* Computers and Operations Research, 2012. Accepted. (Cited on pages 3 and 12.)

[Beasley 2009] J E Beasley. *OR-Library*, July 2009. `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`. (Cited on pages 26 and 28.)

[Belov 2002] G. Belov and G. Scheithauer. *A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths.* European Journal of Operational Research, vol. 141, no. 2, pages 274 – 294, 2002. (Cited on pages 15 and 19.)

[Ben-Tal 2002] A. Ben-Tal and A. Nemirovski. *Robust optimization – methodology and applications.* Mathematical Programming, vol. 92, pages 453–480, 2002. (Cited on pages 73 and 110.)

[Bidot 2009] Julien Bidot, Thierry Vidal, Philippe Laborie and J. Beck. *A theoretic and practical framework for scheduling in a stochastic environment.* Journal of Scheduling, vol. 12, pages 315–344, 2009. 10.1007/s10951-008-0080-x. (Cited on pages 8 and 80.)

[Bisaillon 2009] S. Bisaillon, J.F. Cordeau, G. Laporte and F. Pasin. *A large neighbourhood search heuristic for the aircraft and passenger recovery problem.* 4OR: A Quarterly Journal of Operations Research, pages 1–19, 2009. (Cited on page 42.)

[Blum 2008] Christian Blum, Maria J. Blesa, Andrea Roli and Michael Sampels, editeurs. Hybrid metaheuristics, volume 114 of *Studies in Computational Intelligence*. Springer, 2008. (Cited on page 42.)

[Caprara 2004] Alberto Caprara and Ulrich Pferschy. *Worst-case analysis of the subset sum algorithm for bin packing.* Operations Research Letters, vol. 32, no. 2, pages 159 – 166, 2004. (Cited on pages 17 and 23.)

[Caprara 2005] Alberto Caprara and Ulrich Pferschy. *Modified subset sum heuristics for bin packing.* Information Processing Letters, vol. 96, no. 1, pages 18 – 23, 2005. (Cited on pages 17 and 23.)

[Carey 1999] M. Carey. *Ex ante heuristic measures of schedule reliability.* Transportation Research Part B: Methodological, vol. 33, no. 7, pages 473–494, 1999. (Cited on page 111.)

[Carey 2000] M. Carey and Carville S. *Testing schedule performance and reliability for train stations.* Journal of the Operational Research Society, vol. 51, no. 6, pages 666–682, 2000. (Cited on page 112.)

[Cherri 2009] Adriana Cristina Cherri, Marcos Nereu Arenales and Horacio Hideki Yanasse. *The one-dimensional cutting stock problem with usable leftover - A heuristic approach.* Eur. J. Oper. Res., vol. 196, no. 3, pages 897 – 908, 2009. (Cited on pages 15, 16 and 35.)

[Chowdhury 2002] Badrul H. Chowdhury and Saifur Rahman. *A review of recent advances in economic dispatch.* Power Systems, IEEE Transactions on, vol. 5, no. 4, pages 1248–1259, 2002. (Cited on page 41.)

[Corman 2011a] F. Corman, A. D'Ariano, D. Pacciarelli and Pranzo M. *Dispatching and coordination in multi-area railway traffic management.* Technical report, Dip. Inform. Autom. - Roma Tre University, 2011. (Cited on pages 110, 117 and 118.)

[Corman 2011b] F. Corman, A. D'Ariano, M. Pranzo and Hansen I.A. *Effectiveness of dynamic reordering and rerouting of trains in a complicated and densely occupied station area.* Transportation Planning and Technology, vol. 34, no. 4, pages 341–362, 2011. (Cited on pages 3, 115, 117, 118 and 134.)

[Corman 2012] F. Corman, A. D'Ariano, D. Pacciarelli and Pranzo M. *Bi-objective conflict detection and resolution in railway traffic management.* Transportation Research Part C: Emerging Technologies, vol. 20, no. 1, pages 79–94, 2012. (Cited on page 111.)

[Correia 2008] Isabel Correia, Luís Gouveia and Francisco Saldanha-da Gama. *Solving the variable size bin packing problem with discretized formulations.* Comput. Oper. Res., vol. 35, no. 6, pages 2103–2113, 2008. (Cited on page 12.)

[Cowling 2002] Peter Cowling and Marcus Johansson. *Using real time information for effective dynamic scheduling.* European Journal of Operational Research, vol. 139, no. 2, pages 230 – 244, 2002. (Cited on page 80.)

[Cui 2010] Yaodong Cui and Yuli Yang. *A heuristic for the one-dimensional cutting stock problem with usable leftover.* Eur. J. Oper. Res., vol. 204, no. 2, pages 245 – 250, 2010. (Cited on pages 15, 16 and 35.)

[D'Ariano 2008] A. D'Ariano, C. Corman, D. Pacciarelli and M. Pranzo. *Modeling reordering and local rerouting strategies to solve train conflicts during rail operations.* Transportation Science, vol. 42, pages 405–419, 2008. (Cited on pages 114 and 118.)

[D'Ariano 2009] Andrea D'Ariano and Marco Pranzo. *An Advanced Real-Time Train Dispatching System for Minimizing the Propagation of Delays in a Dispatching Area Under Severe Disturbances.* Networks and Spatial Economics, vol. 9, no. 1, pages 63–84, March 2009. (Cited on page 114.)

[Dewilde 2011] T. Dewilde, P. Sels, D. Cattrysse and P. Vansteenwegen. *Defining Robustness of a Railway Timetable.* Proceedings of the 4th International Seminar on Railway Operations Modelling and Analysis 2011, IAROR 2011, pages 1–20, 2011. (Cited on page 111.)

[Dimitriadis 2009] Sotirios Dimitriadis and Evangelos Kehris. *Cutting stock process optimization in custom door and window manufacturing industry.* International Journal of Decision Sciences, Risk and Management, vol. 1, no. 1 – 2, pages 66 – 80, 2009. (Cited on page 16.)

[Dunning 2001] Dennis J. Dunning, Steve Lockfort, Quentin E. Ross, Phillip C. Beccue and Jeffrey S. Stonebraker. *New York Power Authority uses decision analysis to schedule refueling of its Indian Point 3 nuclear power plant.* Interfaces, vol. 31, no. 5, pages 121–135, 2001. (Cited on page 41.)

[Dyckhoff 1990] Harald Dyckhoff. *A typology of cutting and packing problems.* Eur. J. Oper. Res., vol. 44, no. 2, pages 145 – 159, 1990. (Cited on page 14.)

[Ehrgott 2000] Matthias Ehrgott and Xavier Gandibleux. *A survey and annotated bibliography of multiobjective combinatorial optimization.* OR Spectrum, vol. 22, pages 425–460, 2000. 10.1007/s002910000046. (Cited on page 8.)

[Elmaghraby 1995] Salah E. Elmaghraby. *Activity nets: A guided tour through some recent developments.* European Journal of Operational Research, vol. 82, pages 383 – 408, 1995. (Cited on pages 8 and 77.)

[Falkenauer 1992] E. Falkenauer and A Delchambre. *A genetic algorithm for bin packing and line balancing.* In Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on, pages 1186–1192, 1992. (Cited on pages 16, 26 and 28.)

[Fischetti 2009a] M. Fischetti and Monaci M. *Light Robustness.* Lecture Notes in Computer Science, vol. 5868, pages 61–84, 2009. (Cited on page 111.)

[Fischetti 2009b] M. Fischetti, D. Salvagnin and Zanette A. *Fast approaches to improve the robustness of a railway timetable.* Transportation Science, vol. 43, no. 3, pages 321–335, 2009. (Cited on page 111.)

[Fleszar 2002] Krzysztof Fleszar and Khalil S. Hindi. *New heuristics for one-dimensional bin-packing.* Comput. Oper. Res., vol. 29, no. 7, pages 821 – 839, 2002. (Cited on pages 17, 18 and 26.)

[Fourcade 1997] Fabrice Fourcade, Ellis Johnson, Mourad Bara and Philippe Cortey-Dumont. *Optimizing nuclear power plant refueling with mixed-integer programming.* European Journal of Operational Research, vol. 97, no. 2, pages 269–280, 1997. (Cited on page 40.)

[Gilmore 1961] P.C. Gilmore and R.E. Gomory. *A LINEAR PROGRAMMING APPROACH TO THE CUTTING-STOCK PROBLEM.* Oper. Res., vol. 9, no. 6, pages 849 – 859, 1961. (Cited on page 12.)

[Ginkel 2007] A. Ginkel and Schöbel A. *To wait or not to wait? The bicriteria delay management problem in public transportation.* Transportation Science, vol. 41, no. 4, pages 527–538, 2007. (Cited on page 111.)

[Godskesen 2011] S Godskesen, T Jensen, N Kjeldsen and R Larsen. *Solving a real-life large-scale energy management problem.* 2011. Submitted *Journal of Scheduling.* (Cited on pages 3 and 38.)

[Hansen 2008] I.A. Hansen and J. Pachl. Railway Timetable & Traffic: Analysis - Modelling - Simulation. Eurailpress, 2008. (Cited on page 110.)

[Haouari 2009a] Mohamed Haouari and Mehdi Serairi. *Heuristics for the variable sized bin-packing problem.* Comput. Oper. Res., vol. 36, no. 10, pages 2877–2884, 2009. (Cited on pages 15, 17, 18, 26, 28 and 35.)

[Haouari 2009b] Mohamed Haouari and Mehdi Serairi. *Relaxations and exact solution of the variable sized bin packing problem.* Computational Optimization and Applications, pages 1–24, 2009. (Cited on pages 15 and 19.)

[Honkomp 1999] S.J. Honkomp, L. Mockus and G.V. Reklaitis. *A framework for schedule evaluation with processing uncertainty.* Computers & Chemical Engineering, vol. 23, no. 4 - 5, pages 595 – 609, 1999. (Cited on page 80.)

[Hooker 2011] John N. Hooker. *Hybrid Modeling.* In Panos M. Pardalos, Pascal van Hentenryck and Michela Milano, editeurs, Hybrid Optimization, volume 45 of *Optimization and Its Applications*, pages 11–62. Springer New York, 2011. (Cited on page 42.)

[Hoos 2004] Holger Hoos and Thomas Stützle. Stochastic local search: Foundations & applications. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. (Cited on page 8.)

[Hutter 2009] Frank Hutter, Holger H. Hoos, Kevin Leyton-brown and Thomas Stützle. *ParamILS: An automatic algorithm configuration framework.* Technical report, 2009. (Cited on page 133.)

[Jensen 2011] Thomas Sejr Jensen. *Application of Metaheuristics to Real-life Scheduling Problems.* PhD in computer science, University of Southern Denmark, 2011. (Cited on page 8.)

[Kellerer 2004] H. Kellerer, U. Pferschy and D. Pisinger. Knapsack problems. Springer, Berlin, Germany, 2004. (Cited on pages 13 and 17.)

[Kirkpatrick 1983] S. Kirkpatrick, C. Daniel Gelatt and Mario P. Vecchi. *Optimization by simulated annealing.* Science, vol. 220, no. 4598, page 671, 1983. (Cited on page 56.)

[Koch 2009] Sören Koch, Sebastian König and Gerhard Wäscher. *Integer linear programming for a cutting problem in the wood-processing industry: a case study.* International Transactions in Operational Research, vol. 16, no. 6, pages 715–726, 2009. (Cited on page 16.)

[Laborie 2003] Philippe Laborie. *Resource Temporal Networks: Definition and Complexity.* In IJCAI, pages 948–953, 2003. (Cited on page 8.)

[Larsen 2008] R Larsen. Optimizing Palletflow and Sampling Sequences in Connection with Retrieval of Goods from a Warehouse. Master's thesis, University of Southern Denmark, Odense, 2008. (Cited on pages 9, 72, 105 and 107.)

[Larsen 2012] R. Larsen and M. Pranzo. *A framework for dynamic rescheduling problems.* Technical report, Dip. Ingengeria dell'Informazione – University of Siena, 2012. (Cited on pages 3, 72 and 110.)

[Liebchen 2009] C. Liebchen, M. Lübbecke, R. Möhring and Stiller S. *The concept of recoverable robustness, linear programming recovery, and railway applications.* Lecture Notes in Computer Science, vol. 5868, pages 1–27, 2009. (Cited on page 111.)

[Liebchen 2010] C. Liebchen, M. Schachtebeck, A. Schöbel, S. Stiller and Prigge A. *Computing delay resistant railway timetables.* Computers & Operations Research, vol. 37, pages 857–868, 2010. (Cited on page 111.)

[Lusby 2011] R. Lusby, J. Larsen, M. Ehrgott and Ryan D. *Railway track allocation: models and methods.* OR Spectrum, vol. 33, pages 843–883, 2011. (Cited on page 110.)

[Maccarthy 1993] B.L. Maccarthy and Jiyin Liu. *Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling.* International Journal of Production Research, vol. 31, no. 1, pages 59 – 79, 1993. (Cited on page 72.)

[Mascis 2002] Alessandro Mascis and Dario Pacciarelli. *Job-shop scheduling with blocking and no-wait constraints.* European Journal of Operational Research, vol. 143, no. 3, pages 498 – 517, 2002. (Cited on pages 8, 9 and 77.)

[McKay 1999] Kenneth N. McKay and Vincent C.S. Wiers. *Unifying the Theory and Practice of Production Scheduling.* Journal of Manufacturing Systems, vol. 18, no. 4, pages 241 – 255, 1999. (Cited on page 72.)

[Meng 2011] Lingyun Meng and Xuesong Zhou. *Robust single-track train dispatching model under a dynamic and stochastic environment: A scenario-based rolling horizon solution approach.* Transportation Research Part B: Methodological, vol. 45, pages 1080–1102, 2011. (Cited on page 112.)

[Muth 1963] J.F. Muth and G.L. Thompson (eds). Industrial scheduling. Kluver Academic, Amsterdam, 1963. (Cited on page 96.)

[Neumann 2003] Klaus Neumann, Christoph Schwindt and Jürgen Zimmermann. Project scheduling with time windows and scarce resources. Springer, 2003. (Cited on page 41.)

[Ouelhadj 2009] Djamila Ouelhadj and Sanja Petrovic. *A survey of dynamic scheduling in manufacturing systems.* J. of Scheduling, vol. 12, no. 4, pages 417–431, August 2009. (Cited on pages 5, 6, 75, 79 and 110.)

[Pfeiffer 2007] András Pfeiffer, Botond Kádár and László Monostori. *Stability-oriented evaluation of rescheduling strategies, by using simulation.* Computers in Industry, vol. 58, no. 7, pages 630 – 643, 2007. (Cited on page 80.)

[Pferschy 1999] U. Pferschy. *Dynamic Programming Revisited: Improving Knapsack Algorithms.* Computing, vol. 63, pages 419–430, 1999. (Cited on page 17.)

[Pinedo 1995] M. Pinedo. Scheduling – theory, algorithms and systems. int. series in industrial and system engineering. Prentice-Hall, Englewood Cliffs, NJ, 1995. (Cited on page 95.)

[Pinedo 2009] M. Pinedo. Planning and scheduling in manufacturing and services. Springer, 2009. (Cited on page 5.)

[Pisinger 1999] David Pisinger. *Linear Time Algorithms for Knapsack Problems with Bounded Weights*. Journal of Algorithms, vol. 33, no. 1, pages 1 – 14, 1999. (Cited on page 17.)

[Poldi 2009] Kelly Cristina Poldi and Marcos Nereu Arenales. *Heuristics for the one-dimensional cutting stock problem with limited multiple stock lengths*. Comput. Oper. Res., vol. 36, no. 6, pages 2074 – 2081, 2009. (Cited on pages 26, 27, 28, 29 and 35.)

[Porcheron 2010] Marc Porcheron, Agnès George, Olivier Juan, Tomas Simovic and Guillaume Dereu. *Challenge ROADEF/EURO 2010: A large-scale energy management problem with varied constraints*. `http://challenge.roadef.org/2010/index.en.htm`, February 2010. (Cited on pages 39, 43, 46, 64 and 65.)

[Pranzo 2003] M. Pranzo, C. Meloni and Pacciarelli D. *A new class of greedy heuristics for job shop scheduling problems*. Lecture Notes in Computer Science, vol. 2647, pages 223–236, 2003. (Cited on page 118.)

[Ramasesh 1990] R Ramasesh. *Dynamic job shop scheduling: A survey of simulation research*. Omega, vol. 18, no. 1, pages 43 – 57, 1990. (Cited on page 79.)

[Rasconi 2010] Riccardo Rasconi, Amedeo Cesta and Nicola Policella. *Validating scheduling approaches against executional uncertainty*. Journal of Intelligent Manufacturing, vol. 21, pages 49–64, 2010. 10.1007/s10845-008-0172-7. (Cited on page 80.)

[Roy 1964] B. Roy and B. Sussman. *Les problèm d'ordonnancement avec contraintes disjonctives*. Technical report, Note DS No. 9bis. Paris: SEMA, 1964. (Cited on pages 8 and 77.)

[Ruszczynski 2003] A. Ruszczynski and A. Shapiro. Stochastic programming. in: Handbooks in operations research and management science. Elsevier, Amsterdam, 2003. (Cited on pages 73 and 110.)

[Salido 2008] M.A. Salido, F. Barber and Ingolotti L. *Robustness in railway transportation scheduling*. Proceedings of the World Congress on Intelligent Control and Automation (WCICA), pages 2833–2837, 2008. (Cited on pages 111 and 112.)

[Schaefer 1978] T.J. Schaefer. *The complexity of satisfiability problems*. In Proceedings of the tenth annual ACM symposium on Theory of computing, pages 216–226. ACM, 1978. (Cited on page 47.)

[Schöbel 2009] A. Schöbel and Kratz A. *A Bicriteria Approach for Robust Timetabling.* Lecture Notes in Computer Science, vol. 5868, pages 119–144, 2009. (Cited on page 111.)

[Schulte 2010] Christian Schulte, Guido Tack and Mikael Z. Lagerkvist. *Modeling and Programming with Gecode.* http://www.gecode.org/doc-latest/MPG.pdf, 2010. (Cited on page 65.)

[Schöbel 2006] Anita Schöbel. Optimization in public transportation. Springer, 2006. (Cited on page 5.)

[Shafia 2012] M.A. Shafia, M. Pourseyed Aghaee, S.J. Sadjadi and Jamili A. *Robust train timetabling problem: Mathematical model and branch and bound algorithm.* IEEE Transactions on Intelligent Transportation Systems, vol. 13, pages 307–317, 2012. (Cited on page 112.)

[Skov 2007] J Skov. Scheduling of an Anodizing Plant at Bang & Olufsen. Master's thesis, University of Southern Denmark, Odense, 2007. (Cited on pages 105 and 133.)

[Smith 1956] Wayne E. Smith. *Various optimizers for single-stage production.* Naval Research Logistics Quarterly, vol. 3, no. 1-2, pages 59–66, 1956. (Cited on page 91.)

[Takeuchi 2007] Y. Takeuchi, N. Tomii and Hirai C. *Evaluation method of robustness for train schedules.* Quarterly Report of RTRI (Railway Technical Research Institute) (Japan), vol. 48, no. 4, pages 197–201, 2007. (Cited on page 112.)

[van Hentenryck 2011] Pascal van Hentenryck and Michela Milano. Hybrid optimization, the ten years of cpaior, volume 45 of *Optimization and its applications.* Springer, 2011. (Cited on page 42.)

[Vansteenwegen 2006] P. Vansteenwegen and Van Oudheusden D. *Developing railway timetables which guarantee a better service.* European Journal of Operational Research, vol. 173, pages 337–350, 2006. (Cited on page 112.)

[Vieira 2003] Guilherme E. Vieira, Jeffrey W. Herrmann and Edward Lin. *Rescheduling Manufacturing Systems: A Framework of Strategies, Policies, and Methods.* Journal of Scheduling, vol. 6, pages 39–62, 2003. 10.1023/A:1022235519958. (Cited on page 79.)

[Wäscher 2007] Gerhard Wäscher, Heike Haußner and Holger Schumann. *An improved typology of cutting and packing problems.* Eur. J. Oper. Res., vol. 183, no. 3, pages 1109 – 1130, 2007. (Cited on pages 12, 14 and 33.)

[Yuan 2006] J. Yuan. *Stochastic Modelling of Train Delays and Delay Propagation in Stations.* TRAIL Thesis Series T2006/6, The Netherlands, 2006. (Cited on page 118.)

[Zhang 2000] Guochuan Zhang, Xiaoqiang Cai and C. K. Wong. *Linear time-approximation algorithms for bin packing.* Oper. Res. Lett., vol. 26, no. 5, pages 217 – 222, 2000. (Cited on pages 18 and 26.)

List of Abbreviations

AMCC A greedy heuristic based on the alternative graph model.

B&B Branch and bound.

BPP Classical bin packing problem.

Con. delay Consecutive delay.

CP Constraint programming.

DPLS Dynamic programming based local search.

DPLSW Dynamic programming based local search ignoring a set amount of waste.

FFD First fit decreasing.

FIFO First in first out.

JSP Job shop scheduling problem.

Late. Lateness.

LP Linear programming.

MIP Mixed integer programming.

RCPSP Resource constrained project scheduling problem.

SLS Stochastic local search.

SMWCTP Single Machine Weighted Completion Time Problem.

VSBPP Variable size bin packing problem.

WSPT Weighted Shortest Processing Time