Algorithmic Developments in Two-Stage Robust Scheduling

Matthew Bold, B.Sc.(Hons.), M.Res

Submitted for the degree of Doctor of Philosophy at Lancaster University.

June 2022

Abstract

This thesis considers the modelling and solving of a range of scheduling problems, with a particular focus on the use of robust optimisation for scheduling in two-stage decision-making contexts.

One key contribution of this thesis is the development of a new compact robust counterpart for the resource-constrained project scheduling problem with uncertain activity durations. Resource conflicts must be resolved under the assumption of budgeted uncertainty, but start times can be determined once the activity durations become known. This formulation is also applied to the multi-mode version of this problem. In both cases, computational results show the clear dominance of the new formulation over the prior decomposition-based state-of-the-art methods.

This thesis also demonstrates the first application of the recoverable robust framework to single machine scheduling. Two variants of this problem are considered, in which a first-stage schedule is constructed subject to uncertain job processing times, but can be amended in some limited way following the realisation of these processing times. The first of these problems is considered under general polyhedral uncertainty. Key results concerning the second-stage subproblem are derived, resulting in three formulations to the full problem which are compared computationally. The second of these problems considers interval uncertainty but allows for a more general recovery action. A 2-approximation is derived and the performance of a proposed greedy algorithm is examined in a series of computational experiments.

In addition to these results on two-stage robust scheduling problems, a new deterministic resource-constrained project scheduling model is developed which, for the first time, combines both generalised precedence constraints and flexible resource allocation. This model is introduced specifically for the application of scheduling the decommissioning of the Sellafield nuclear site. A genetic algorithm is proposed to solve this model, and its performance is compared against a mixed-integer programming formulation.

Acknowledgements

First and foremost, I would like to thank my supervisors, Marc Goerigk, Burak Boyaci and Chris Kirkbride. I certainly wouldn't be here without their patience, encouragement, enthusiasm and good ideas. They have taught me a huge amount and I will greatly miss working with them.

Thank you also to Panos Frangos, Andrew MacPherson and Steve Hastewell from Sellafield Ltd for being so generous with their time and their feedback, as well as for their exhaustive data collection efforts throughout the project.

I feel particularly lucky to have been a part of the STOR-i Centre for Doctoral training. Big thanks to the people that have helped to run STOR-i (Jon, Kevin, Idris, Jen, Wendy, Kim, Nicky, Oli, Dan, Georgie), and cheers to all of the other wonderful students of STOR-i for making it such a fun and memorable few years. I am, of course, also thankful to the EPSRC for funding my time at STOR-i.

Big love to Tom, Alan and Mirjam. Thanks for all the great times together at 110 Bowerham Road. Huge thanks also goes to Em, for her words of wisdom and encouragement and for so many moments of joy. Finally, I would like to thank Mum and Dad and the rest of my family for their endless support, interest and pride. It means a lot.

Declaration

I declare that the work in this thesis has been done by myself in collaboration with my supervisors, and has not been submitted elsewhere for the award of any other degree.

The contents of Chapter 5 has been published as Bold, M. and Goerigk, M. (2021). A compact reformulation of the two-stage robust resource-constrained project scheduling problem. *Computers & Operations Research*, 130:105232.

The contents of Chapter 6 has been submitted for publication as Bold, M., Goerigk, M. (2022). A faster exact method for solving the robust multi-mode resource-constrained project scheduling problem.

The contents of Chapter 7 has been submitted for publication as Bold, M., Goerigk, M. (2022). Recoverable Robust Single Machine Scheduling with Polyhedral Uncertainty. (We note that this work is awaiting feedback from the *Journal of Scheduling* following revisions made in response to a first set of reviews.)

The contents of Chapter 8 has been published as Bold, M. and Goerigk, M. (2022). Investigating the recoverable robust single machine scheduling problem under interval uncertainty. *Discrete Applied Mathematics*, 313:99-114.

Matthew Bold

Contents

List of Figures

List of Tables

For Grandad

Chapter 1

Introduction

Scheduling has been one of the most active areas of operational research since the mid-1950s, and continues to be very well-studied to this day. This comes as no surprise given the broad range of fields for which scheduling is of critical importance, including construction, engineering, manufacturing, transportation, and computer science, to name just a few. Indeed, it is estimated that in 2020, Advanced Planning and Scheduling (APS) software had a total global market value of \$1.5 billion[1].

The work in this thesis contributes to this active area of research by developing and analysing a set of novel models and algorithms for a number of different scheduling problems, with a particular focus on the application of robust optimisation to problems with uncertain activity durations. The scheduling problems considered in this thesis have each, to a greater or lesser extent, been motivated by the specific challenges involved in the scheduling of the decommissioning of the

[1] https://www.statista.com/statistics/1238886/worldwide-advanced-planning-scheduling-market/. Accessed: 29/03/2022

Sellafield nuclear site.

Covering 6 square kilometers on the Cumbrian coast, the Sellafield site contains more than 200 legacy nuclear facilities which together, house the world's largest inventory of untreated nuclear waste. The systematic dismantling of these facilities and remediation of the site is a vast project, expected to take in excess of 100 years to complete and cost over £90 billion (NDA, 2019). Given its scale and complexity, it is crucial that this program is scheduled carefully and systematically in a manner that accounts for all of the features and constraints of the site. In Chapter 4, a model is specifically developed for scheduling the decommissioning of the site.

There exists a large degree of uncertainty surrounding the quantity and condition of much of the waste contained within the oldest facilities on the Sellafield site. When it comes to scheduling the decommissioning of the site, this uncertainty primarily manifests itself as durational uncertainty in the decommissioning activities. Therefore the development of scheduling approaches that are able to account for this durational uncertainty are of significant interest to Sellafield. The later chapters consider a range of scheduling problems, and are linked by their use of robust optimisation to account for uncertainty in the durations of activities. So although these later chapters do not directly reference Sellafield decommissioning, their shared direction of research have been guided by this particular challenge. Furthermore, the problems in these chapters also share a two-stage decision framework in which a first-stage decision is made under the problem uncertainty, but can be extended or amended once that uncertainty has been resolved in the second-stage.

The problems considered in this thesis can also be broadly grouped by the following two categories: 1.resource-constrained project scheduling problems, and 2. single machine scheduling problems. Resource-constrained project scheduling, as considered by Chapters 4, 5 and 6, is concerned with the assignment of multiple types of scarce resource to a set of project activities which are related through a network of logical precedence relationships. Single machine scheduling, on the other hand, involves the ordering of a set of jobs on a single machine, with the aim of optimising some objective. In Chapters 7 and 8 we consider two versions of this problem with the objective of minimising the sum of job completion times, subject to uncertainty.

The content and of contributions of each of the chapters of this thesis are outlined below.

- **Chapter 2: Deterministic Resource-Constrained Project Scheduling.** This chapter provides an overview of the most relevant existing literature that relates to the classic model for project scheduling known as the resource-constrained project scheduling problem (RCPSP). A number of extensions to this model that are considered in this thesis are also covered.

- **Chapter 3: Robust Optimisation.** This chapter serves to introduce the framework of robust optimisation. Particular attention is paid to developments in two-stage robust combinatorial optimisation with the aim of providing the necessary context for the work in Chapters 5-8.

- **Chapter 4: The Generalised Flexible Resource-Constrained Project Scheduling Problem.** Motivated by the real-world Sellafield decommissioning problem, this chapter introduces a novel extension to the deterministic RCPSP that includes both generalised precedence constraints and flexible resource allocation. This new problem is referred to as the generalised flexible resource-constrained project scheduling problem (GFRCPSP). A mixed-integer linear programming (MILP) formulation and genetic algorithm are proposed to solve this problem, and a computational study demonstrates the effectiveness of the genetic algorithm. The newly developed GFRCPSP scheduling model is used to solve a version of the Sellafield decommissioning problem.

A shortened version of this work was presented at the *17th International Workshop on Project Management and Scheduling*, 2021.

- **Chapter 5: A Compact Reformulation of the Two-Stage Robust Resource-Constrained Project Scheduling Problem.** This chapter considers the RCPSP with uncertain activity durations. The proposed approach makes use of a two-stage decision process and assumes that activity durations lie in a budgeted uncertainty set. A new reformulation of the second-stage problem is introduced, which enables the derivation of the first compact robust counterpart to the full two-stage adjustable robust optimisation problem. Computational experiments show that this compact formulation can be solved using standard optimisation software significantly faster than the prior decompostion-based state-of-the-art algorithm for solving this problem, reaching optimality for almost 50% more instances on the same benchmark set.

  This work has been published as Bold, M. and Goerigk, M. (2021). A compact reformulation of the two-stage robust resource-constrained project scheduling problem. *Computers & Operations Research*, 130:105232.

- **Chapter 6: A Faster Exact Method for Solving the Robust Multi-Mode Resource-Constrained Project Scheduling Problem.** The chapter extends the formulation developed in Chapter 5 for application to the two-stage robust multi-mode resource-constrained project scheduling problem. Results from this formulation are shown to dominate an improved version of the prior state-of-the-art method for solving this problem.

  This work has been submitted for publication as Bold, M., Goerigk, M. (2022). A faster exact method for solving the robust multi-mode resource-constrained project scheduling problem.

- **Chapter 7: Recoverable Robust Single Machine Scheduling with Polyhedral Uncertainty.** This chapter considers a recoverable robust single machine scheduling problem under general polyhedral uncertainty with the objective of minimising the total sum of completion times. A specific recovery framework is considered in which up to Δ distinct pairs of jobs can be swapped when amending the first-stage schedule. Following its formulation using a general recoverable robust framework, the structure of this problem is examined in detail to reveal a positive complexity result for matching problems with a particular cost structure, resulting in the development of a further two formulations for this problem. The relative strengths of these models are compared computationally for the case of budgeted uncertainty.

  This work has been submitted for publication as Bold, M., Goerigk, M. (2022). Recoverable Robust Single Machine Scheduling with Polyhedral Uncertainty.

- **Chapter 8: Investigating the Recoverable Robust Single Machine Scheduling Problem Under Interval Uncertainty.** This chapter also considers the recoverable robust single machine scheduling problem with the objective of minimising the total sum of completion times, but does so in the context of interval uncertainty and a more general recovery action, which specifies that at least Δ jobs share the same position in the first and second-stage schedules. A handful of important special cases of this problem are shown to be polynomially solvable, before a 2-approximation algorithm is derived for the full problem. Computational experiments examine the performance of an exact MILP formulation and the approximation algorithm, and demonstrate the strength of a proposed polynomial-time greedy heuristic.

  This work has been published as Bold, M. and Goerigk, M. (2022). Investigating the recoverable robust single machine scheduling problem under interval uncertainty. *Discrete Applied Mathematics*, 313:99-114.

- **Chapter 9: Conclusions.** Concluding remarks are made, and a number of possible suggestions for building on the work in the thesis are given.

As noted in the summaries above, Chapters 5, 6, 7 and 8 have each been have been either published or are currently in submission as standalone papers. These have been included here in their published or submitted form, with only very minor edits to frame them into the context of the thesis. These chapters can each, therefore, be read as self-contained pieces of work and as a result, may contain material that overlaps to some degree with each another and with the literature reviews presented in Chapters 2 and 3.

Chapter 2

Deterministic Resource-Constrained Project Scheduling

This chapter surveys the most relevant existing literature regarding the deterministic resource-constrained project scheduling problem (RCPSP), as well as the extensions and variants of this problem that are also considered in the work of this thesis.

Before beginning this review, we make note of the numerous survey papers (Herroelen et al., 1998; Brucker et al., 1999; Kolisch and Padman, 2001; Hartmann and Briskorn, 2010, 2022) and books (Demeulemeester and Herroelen, 2006; Weglarz, 2012; Artigues et al., 2013a; Schwindt et al., 2015) on the RCPSP and deterministic project scheduling more generally, many of which have aided the writing of this chapter.

2.1 The RCPSP

Stated briefly, the RCPSP consists of finding a feasible project schedule that minimises the overall project completion time. Given its numerous applications throughout industry, as well as its theoretical interest, the RCPSP has been one of the most widely-studied problems in operational research since a model was first proposed by Pritsker et al. (1969).

A project consists of a set of activities $V = \{0, 1, \ldots, n, n+1\}$, where 0 and $n+1$ are dummy activities that represent the start and end of the project, respectively. Each activity $i \in V$ has a duration d_i and a requirement r_{ik} of each resource type $k \in K$. Each resource is assumed to be *renewable*, meaning there is an constant availability of R_k in each time period of the project scheduling horizon T. The project activities are related through a set of strict finish-to-start precedence relationships. These can be represented on a directed acyclic graph $G = (V, E)$, where arcs E capture the relationships between the activities V. The activities are also assumed to be *non-preemptive*, meaning they must be completed in a single processing phase. The aim of the RCPSP is to determine a resource and precedence feasible schedule, i.e. a start time for each activity $i \in V$, that minimises the overall project duration, known as the *makespan*. An example RCPSP instance is shown in Figure 2.1 and its optimal solution is given in Figure 2.2.

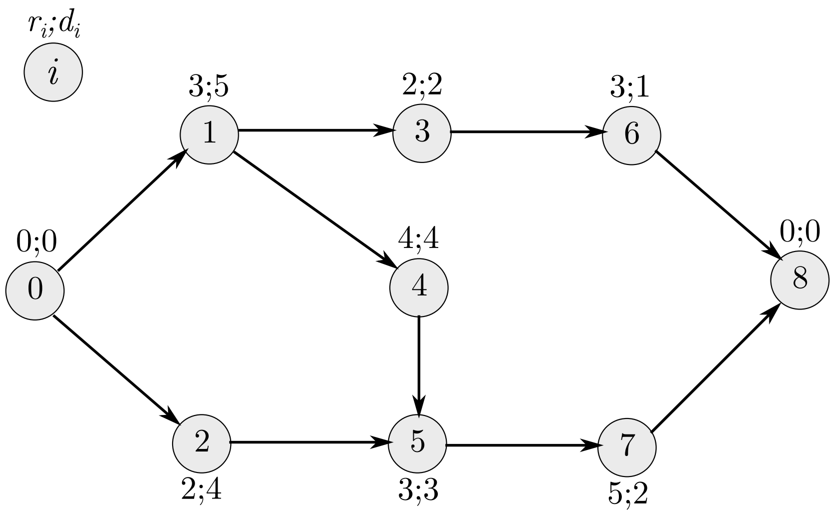


Figure 2.1: An example RCPSP instance involving seven non-dummy activities and a single renewable resource with $R_1 = 6$.

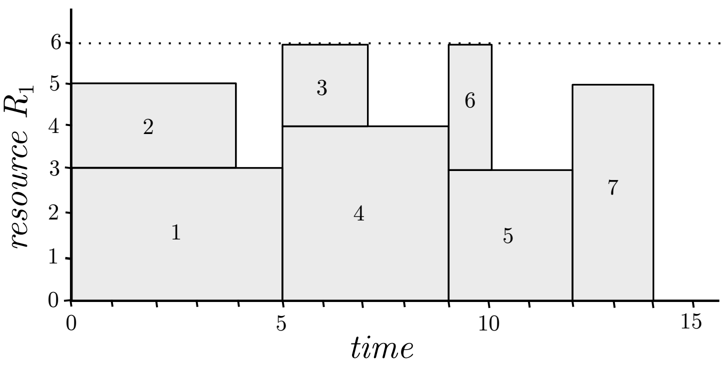


Figure 2.2: An optimal solution to the RCPSP instance shown in Figure 2.1.

Blazewicz et al. (1983) showed that the RCPSP is strongly NP-hard. As a result, exact mixed-integer programming (MILP) formulations can typically only be solved for moderately-sized instances. However, its severe complexity is one of the reasons the RCPSP has garnered so much academic attention, with it having become a testing ground for many general metaheuristic optimisation approaches. Before reviewing some of these approaches, we first present a couple of MILP formulations.

2.1.1 Mixed-integer linear programming formulations

Although there exist many alternative formulations, here we detail only the two most relevant to our work. The first is the original formulation proposed by Pritsker et al. (1969):

$$\min \sum_{t\in T} t x_{n+1,t} \tag{2.1}$$

$$\text{s.t.} \sum_{t\in T} x_{it} = 1 \qquad \forall i \in V \tag{2.2}$$

$$\sum_{t\in T} t x_{jt} \geq \sum_{t\in T} t x_{it} + d_i \qquad \forall (i,j) \in E \tag{2.3}$$

$$\sum_{i\in V} \sum_{\tau=\max\{0,t-d_i+1\}}^{t} r_{ik} x_{i\tau} \leq R_k \qquad \forall k \in K, t \in T \tag{2.4}$$

$$x_{it} \in \{0,1\} \qquad \forall i \in V,\, t \in T, \tag{2.5}$$

where $x_{it} = 1$ if activity i starts in time period t, and $x_{it} = 0$ otherwise. Observe that the activity start times are tracked using $\sum_{t\in T} t x_{it}$ for each $i \in V$. Constraints (2.2) ensure the non-preemption of each activity by allowing allowing only a single start time for each activity, whilst (2.3) and (2.4) enforce the precedence and resource constraints respectively.

This model is referred to as a *discrete-time* formulation, since it defines a set of binary variables for each period in the time horizon. This discrete-time formulation was refined by Christofides et al. (1987) who strengthened the precedence constraints (2.3) by 'disaggregating' them into separate constraints for each of the time periods in the project horizon.

Note that the project precedence constraints enforce a time-window of possible start times for each activity $i \in V$, given by $\{ES_i, \ldots, LS_i\}$, where ES_i and LS_i

denote the earliest and latest start times of activity i. These time-windows can be computed in a simple pre-processing step. By restricting the creation of the x_{it} variables to only the feasible time-windows of each activity, the number of variables required by model (2.1-2.5) can be greatly reduced.

Although this discrete-time model generally produces strong LP relaxations, even when the model variables are pruned using the feasible time-windows of each activity, this formulation involves a pseudo-polynomial number of variables and constraints and therefore suffers particularly poor performance for instances with long time horizons.

Building on the work of Alvarez-Valdès and Tamarit (1993), Artigues et al. (2003) proposed an alternative flow-based continuous-time MILP formulation for the RCPSP. Their formulation makes use of continuous flow variables f_{ijk} to track the amount of resource k that is transferred from activity i to activity j on its route through the network from the dummy source activity 0 to the dummy sink activity $n+1$. Additionally, binary variables x_{ij} indicate whether or not activity i is processed before activity j, and variables S_i define the activity start times. Using these variables this flow-based formulation can be written as

$$\min S_{n+1} \tag{2.6}$$

$$\text{s.t. } x_{ij} = 1 \qquad \forall (i,j) \in E \tag{2.7}$$

$$S_0 = 0 \tag{2.8}$$

$$S_j \geq S_i + d_i - M(1 - x_{ij}) \qquad \forall i,j \in V \tag{2.9}$$

$$f_{ijk} \leq N x_{ij} \qquad \forall i,j \in V,\ k \in K \tag{2.10}$$

$$\sum_{j \in V} f_{ijk} = r_{ik} \qquad \forall i \in V,\ k \in K \tag{2.11}$$

$$\sum_{i \in V} f_{ijk} = r_{jk} \qquad \forall j \in V,\, k \in K \tag{2.12}$$

$$f_{ijk} \geq 0 \qquad \forall i, j, k \in V \tag{2.13}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V, \tag{2.14}$$

where $r_{0k} = r_{n+1,k} = R_k$ for each $k \in K$, and M and N are constants that are large enough to trivially satisfy their corresponding constraints when required. Constraints (2.7) capture the original project precedence constraints. The activity start times are determined by (2.8) and (2.9), which ensure that the start times respect the precedences defined by the x_{ij} variables. Constraints (2.10-2.12) are the resource flow conservation constraints.

The network formed by the flow of resource from source 0 to sink $n + 1$ is an extension of the original project network. As shown by Bartusch et al. (1988), the RCPSP can be reduced to determining the optimal set of additional arcs to include in this extension of the original project network. This observation is based on the consideration of so-called *forbidden sets*. Introduced by Igelmund and Radermacher (1983a,b), a forbidden set $F \subseteq V$ is defined to be any set of activities that are not precedence-related, such that $\sum_{i \in F} r_{ik} > R_k$ for some $k \in K$. That is, a forbidden set is a set of activities that cannot be executed in parallel *only* because they would violate resource constraints. A *minimal forbidden set* is a forbidden set with no subsets that are also forbidden sets. Bartusch et al. (1988) show that the RCPSP is equivalent to the problem of optimally selecting a set $X \subseteq V^2 \setminus E$ of additional precedence constraints, known as a *sufficient selection*, such that $G(V, E \cup X)$ is acyclic and contains no minimal forbidden sets. In a solution to formulation (2.6)-(2.14), the x_{ij} variables exactly define a set of additional precedence constraints X which resolve the resource conflicts, and the flow of resource, given by the f_{ijk} variables, is directed through the extended network $G(V, E \cup X)$. The sufficient selection corresponding to the optimal solution shown in Figure 2.2 is given by $\{(2,4),(4,6),(6,7)\}$. The resulting extended network and flow of resource is shown in Figure 2.3.

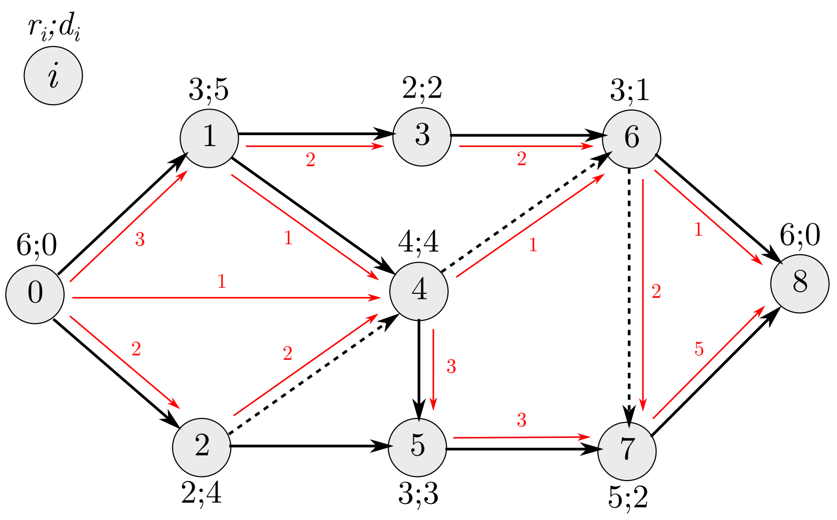


Figure 2.3: Resource flow corresponding to the optimal solution given in Figure 2.2. The sufficient selection given by the dashed arcs, and flow of resource is shown by the arcs in red.

Unlike the time-based formulation, the flow-based formulation contains a polynomial number of variables and constraints. However it yields a weak LP relaxation as a result of its use of Big-M constraints. This formulation is of particular relevance because its separation of the resource allocation and scheduling variables enables the two-stage scheduling approach that is considered in chapters 5 and 6.

A number of other formulations exist for the RCPSP including, notably, a set of event-based continuous-time formulations. For an extensive comparison of MILP formulations for the RCPSP, including the discrete-time and flow-based formulations stated above, see Koné et al. (2011, 2013).

Although we do not cover them in any detail here, we briefly mention that tailored branch and bound (B&B) approaches are typically able to exploit the problem structure much more effectively than the generic solution approaches used in off-the-shelf mathematical solvers. B&B approaches therefore usually obtain better computational results than can be achieved by directly passing a full MILP formulation to one of these solvers. Most early research regarding the RCPSP was concerned with the development of different branching schemes for the incremental extension of the solution schedule. One such scheme is based on the removal of minimal forbidden sets, as described above. The reader is referred to Demeulemeester and Herroelen (2006) for an extensive review of B&B and lower bounding procedures for the RCPSP.

2.1.2 Heuristic approaches

Given its complexity, a large proportion of literature relating to the RCPSP is concerned with the development of scalable heuristic approaches. Here we briefly outline some of the key developments in this regard, focusing particularly on the application of genetic algorithms in preparation of Chapter 4.

Many heuristic scheduling methods are built upon a schedule generation scheme (SGS). An SGS constructs a feasible schedule from a topologically-ordered (with respect to the project precedences) list of the project activities (such a list is referred to as an *activity list*). There exist two types of SGS, the *serial SGS* and the *parallel SGS*. The serial SGS iterates through the activity list and schedules each activity at the earliest precedence and resource feasible time period. On the other hand, the parallel SGS iterates through time periods in the project horizon and schedules as many precedence and resource feasible activities as possible at each time period, doing so in the order dictated by the activity list.

The simplest type of heuristics for the RCPSP are known as priority-rule heuristics. These algorithms simply specify some rule for generating activity lists and decodes them using an SGS. Examples of classic priority-rules include shortest processing time, latest finish time or most total successors. For an analysis of priority-rules for scheduling, see Kolisch (1996a,b). The application of a priority-rule to generate a single schedule is referred to as a single-pass approach. More commonly however, different activity lists and SGS types are combined repeatedly to generate many different schedules in a so-called multi-pass approach. Although priority-rule heuristics are some of the oldest and most basic heuristics for the RCPSP, their simplicity and computational speed mean that they remain widely used, and form the basis of many more complex metaheuristic approaches.

Although countless types of metaheuristics have been proposed for solving the RCPSP including simulated annealing, tabu search, ant colony optimisation and particle swarm optimisation, among others, here we concentrate exclusively on the use of genetic algorithms (GA). This is for two reasons. Firstly, GAs are consistently among the most popular and best performing metaheuristic approaches for the RCPSP, and secondly, a GA is proposed to solve the project scheduling model developed in Chapter 4. For a thorough review of heuristics and metaheuristics for the RCPSP, including a number of genetic algorithms, see Kolisch and Hartmann (1999), Hartmann and Kolisch (2000) and Kolisch and Hartmann (2006).

First proposed by Holland (1975), GAs take their inspiration from the biological process of evolution. They work by strengthening a population of candidate solutions over a course of generations, with new solutions being created by 'breeding' and 'mutating' existing solutions using a so-called *crossover* and *mutation* operators. The best solutions survive to the next generation and gain the chance to spread their characteristics further through the population, whilst the weaker solutions are deleted.

Considering the RCPSP, activity lists provide a very natural encoding on which a GA can operate. Another encoding frequently used in the literature is the *random key representation*, which assigns a real number between 0 and 1 to each project activity that determines its priority in the schedule. As mentioned above, many of the strongest heuristics for solving the RCPSP are GAs. There are too many papers that propose GAs for the RCPSP to exhaustively list them here, so instead we only mention the strongest and most notable of these methods.

Hartmann (1998) proposed a GA based on an activity list encoding, and used the serial SGS to schedule these lists. Their approach employs a so-called *two-point crossover*, which has since become a standard feature of GAs for project scheduling. This works by taking two 'parent' solutions, i.e. a 'mother' and 'father', and choosing two random crossover points. The child solution inherits the activity ordering of its father up to the first point, the activity ordering of its mother between the two points, and the ordering of the remaining activities from its father again after the second point. An example of this crossover is shown in Figure 2.4. Note that a second child can be generated by switching the roles of the two parent solutions. In the first review of heuristic methods for the RCPSP given by Hartmann and Kolisch (2000), this GA was the best performing algorithm. Alcaraz and Maroto (2001) extended this GA to include an extra gene which determines whether the serial SGS is used to do forwards or backwards scheduling. Similarly, Hartmann (2002) strengthened the results from Hartmann (1998) by including a gene to choose between the use of the serial or the parallel SGS. Hindi et al. (2002) also proposed a similar GA to Hartmann (1998), but suggested an alternative strategy for building the initial population of solutions.

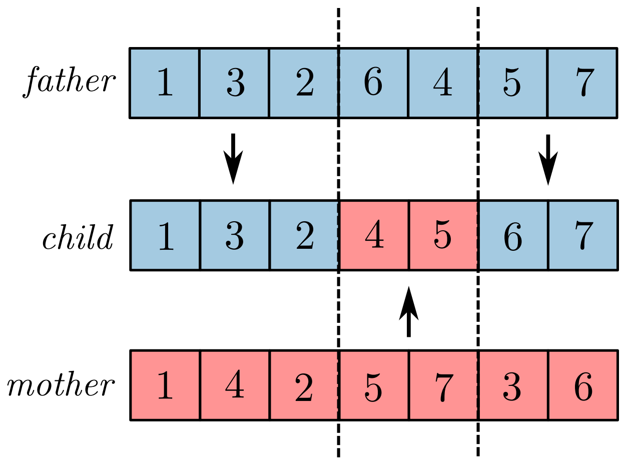


Figure 2.4: Example of the two-point crossover first proposed by Hartmann (1998).

Valls et al. (2005) demonstrate the effectiveness of a forward-backward justification technique that involves right-shifting then left-shifting the project activities to obtain a better schedule. They subsequently employed this technique in a GA in Valls et al. (2008). Their approach remains among the strongest metaheuristic approaches for solving the RCPSP. This work inspired a number of other GAs that also made use of this forward-backward improvement step, including Debels and Vanhoucke (2007), Wang et al. (2010), Gonçalves et al. (2011) and Zamani (2013).

More recent approaches have tended to combine elements from various different metaheuristic approaches in so-called hybrid metaheuristics. Particularly effective examples of hybrid GAs include Proon and Jin (2011), who added a local neighbourhood search to their crossover operator, and Lim et al. (2013), who made use of a simulated-annealing-type search heuristic. Elsayed et al. (2017) propose an algorithm which adapts its use of metaheuristic algorithm depending on its performance on the instance in question. Pellerin et al. (2020) present a computational comparison of an extensive list of hybrid metaheuristic approaches for solving the RCPSP, including those mentioned here.

2.2 The GRCPSP

The basic finish-to-start precedence relationships in the classic RCPSP limit its suitability for modelling many real-world scheduling problems that include features such as processing time windows, fixed start times, setup times, deadlines and minimal and maximal overlap of activities. Given the common occurrence of such constraints, the generalisation of the precedence constraints of the RCPSP is one of the earliest and most widely-studied extensions of this problem. It is worth noting that this extension has been referred to under many different names including the resource-constrained project scheduling problem with time-windows (Bartusch et al., 1988), the resource-constrained project scheduling problem with minimum and maximum time-lags (RCPSP/max) (Neumann and Zhan, 1995), the generalised resource-constrained project scheduling problem (GRCPSP) (Demeulemeester and Herroelen, 1997) and the resource-constrained project scheduling problem with generalised precedence relations (De Reyck and Herroelen, 1998).

Specifically, generalised precedence constraints extend the concept of the basic finish-to-start relationship in the standard RCPSP to include start-to-start, start-to-finish, finish-to-start and finish-to-finish relationships, as well as minimum and maximum time-lags. A minimum time-lag specifies the minimum number of time periods that must elapse between the predecessor and successor, whilst a maximum time-lag specifies the maximum number of time periods that can elapse between the predecessor and successor. The four types of precedence relationships can easily be converted into one another by adjusting the associated time-lag to account for the required addition or subtraction of the relevant activity durations (see Bartusch et al. (1988)). Observe also that maximum time-lag constraints can be converted into negative minimum time-lag constraints. For example consider the constraint $s_j \leq f_i + \ell$, that is, a maximum time-lag of length $\ell \geq 0$ from f_i to s_i, where f_i and s_j denote the finish time of i and the start time of j respectively. This relationship can be rewritten as $f_i \geq s_j - \ell$, which exactly defines a minimum time-lag of length $-\ell$ going from s_j to f_i.

Having pre-processed all the precedence constraints into minimum time-lag start-to-start type, they can be represented on a directed graph as was the case with the RCPSP. As a result of the maximum time-lags however, this graph may contain cycles. A positive length cycle corresponds to a logical inconsistency within the project precedences, e.g. start a before b, and start b before a, and as a result a GRCPSP instance is feasible with respect to its precedence constraints if and only if its precedence network contains no cycles of positive length. This can be checked straightforwardly in $O(|V|^3)$ time by implementing the Floyd-Warshall algorithm to compute a distance matrix containing the longest path between each pair of project activities.

The presence of these cycles make the GRCPSP especially challenging. In particular, a feasible solution to a GRCPSP instance exists if and only if a feasible solution exists to each of the makespan-constrained RCPSP sub-problems corresponding to the separate cycles in the project network. Hence, just determining whether or not a feasible solution exists for a given GRCPSP instance is itself an NP-complete problem (Bartusch et al., 1988).

Figure 2.5 shows an example GRCPSP instance involving the same seven non-dummy activities as the example RCPSP instance from the previous section. This instance contains two temporally-feasible cycles $3 \rightarrow 6 \rightarrow 3$ and $2 \rightarrow 5 \rightarrow 7 \rightarrow 2$. An optimal solution to this instance is shown in Figure 2.6.

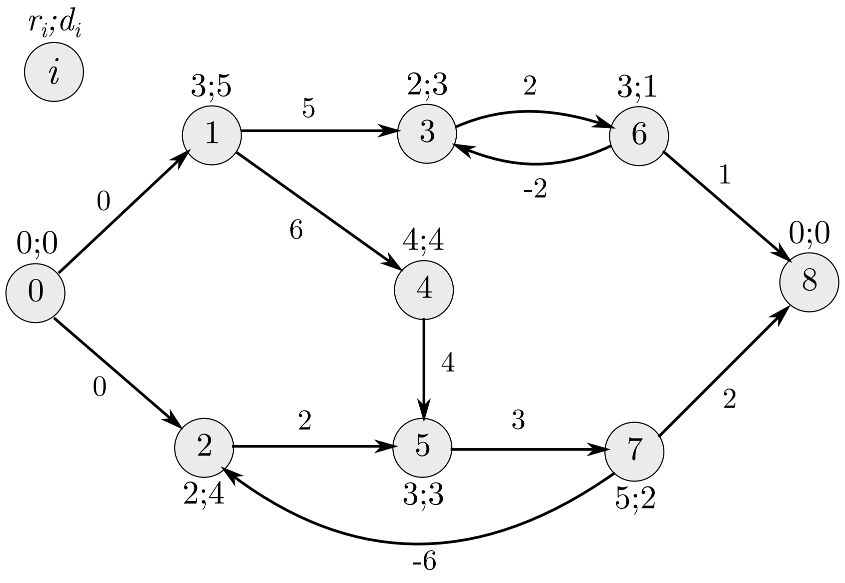


Figure 2.5: An example GRCPSP instance involving the same seven non-dummy activities as the RCPSP instance in Figure 2.1, and one renewable resource with $R_1 = 6$.

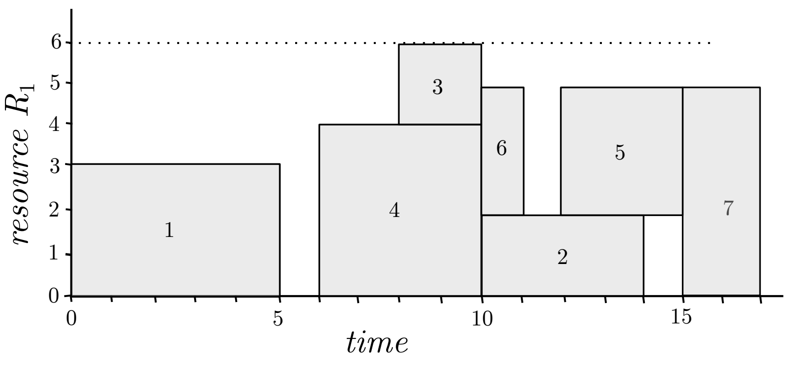


Figure 2.6: An optimal solution to the GRCPSP instance given in Figure 2.5.

Bartusch et al. (1988) were the first to consider this problem and developed a B&B algorithm that branches on the introduction of additional precedence constraints to break forbidden sets. Subsequently, other B&B algorithms also based on this idea were also developed by Demeulemeester and Herroelen (1997) and De Reyck and Herroelen (1998). An alternative time-based B&B approach was presented by Dorndorf et al. (2000), who used constraint propagation to further strengthen its performance. Cesta et al. (2002) proposed a constraint satisfaction procedure that works by incrementally removing resource conflicts from a solution that is precedence feasible. Bianco and Caramia (2011) presented a transformation of the generalised precedence constraints into zero time-lag finish-to-start relationships by introducing dummy activities for each relationship into the original network. Calculating the longest path through this transformed network gives a lower bound to the original problem. Bianco and Caramia (2012) proposed a B&B algorithm based on a new mathematical formulation, and strengthened their algorithm by making use of Lagrangian relaxation. The most effective exact approaches for solving the GRCPSP have been developed by Schutt et al. (2013) and de Azevedo et al. (2021), who both formulate and solve the problem as a constraint satisfaction problem.

A number of heuristic approaches have also been developed to solve the GRCPSP. Neumann and Zhan (1995) were the first to present a number of priority-rule heuristics, and motivated by the decomposition result of Bartusch et al. (1988), they proposed a strategy of generating sub-schedules for each cycle in the project network, before bringing them together to create a full schedule for the original problem. Franck et al. (2001) proposed and compared a number of heuristics for the GRCPSP, including truncated B&B, filter-beam search, priority-rule heuristics, a genetic algorithm and a tabu search. Ballestín et al. (2011) developed an evolutionary algorithm based on a serial SGS and a crossover which attempts to identify and pass-on good orderings of the activities involved in the cycle structures. Their approach is the most effective metaheuristic developed to date for solving the GRCPSP.

2.3 The FRCPSP

The resource-constrained project scheduling problem with flexible resource profiles (FRCPSP) is a more recent and less studied extension to the RCPSP than the GRCPSP. First considered by Kolisch et al. (2003) in the context of pharmaceutical research, the FRCPSP relaxes the assumption that each project activity has a fixed duration and resource requirement, and instead assumes that only the overall amount of resource required to complete each activity is known. The FRCPSP aims to find a resource allocation to each activity at each period of its duration, as well as a start time for each activity, in order to minimise the project makespan. In addition to the usual precedence and resource constraints of the standard RCPSP, a solution to the FRCPSP must also satisfy lower and upper bounds on the per-period allocation of each resource to each activity, as well as a minimum number of periods for which resource allocation to an activity must remain constant, known as the *minimum block length*. An example FRCPSP instance is given in Figure 2.7, and an optimal solution to this instance is shown in Figure 2.8.

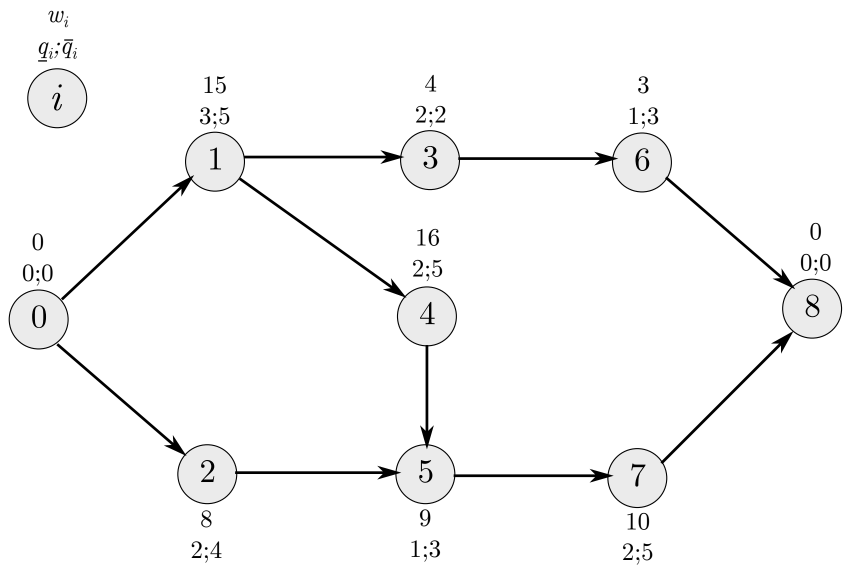


Figure 2.7: An example FRCPSP involving seven non-dummy activities and one renewable resource with $R_1 = 6$. The minimum block length is 2.

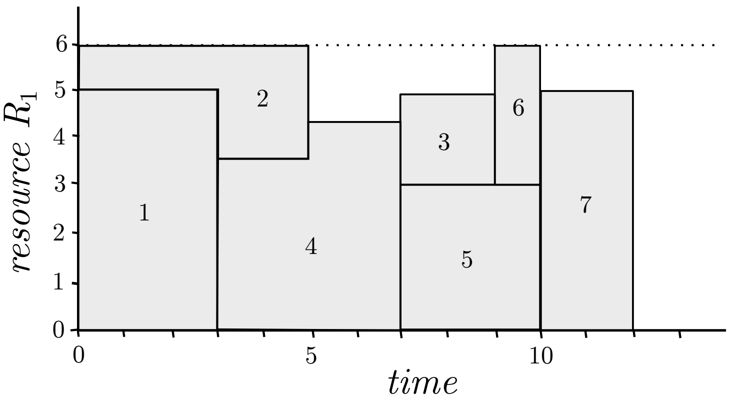


Figure 2.8: An optimal solution to the FRCPSP instance given in Figure 2.7, assuming a minimum block length of 2.

As well as introducing this new scheduling model, Kolisch et al. (2003) proposed an MILP formulation and greedy priority-rule heuristic to solve it. Since then, a handful of other MILP formulations have been presented for this problem. Naber and Kolisch (2014) introduced and compared four discrete-time MILPs, showing the clear superiority of one of them, and more recently, Naber (2017) proposed the first continuous-time MILP formulation for the FRCPSP. Baumann et al. (2015) presented an MILP formulation for the problem with discrete resources.

As with most other project scheduling models, the difficulty of solving large instances with exact methods means that a greater proportion of literature relating to the FRCPSP has been concerned with the development of heuristic solution approaches. For the FRCPSP with discrete resources Fündeling and Trautmann (2010) introduced a priority-rule heuristic based on a greedy serial SGS. Ranjbar and Kianfar (2010) used a GA based on a serial SGS to choose between a limited set of resource allocation profiles for each activity. An alternative approach that uses an MILP-based heuristic to sequentially schedule the project activities was suggested by Zimmermann (2016). Considering the FRCPSP with continuous resources, Schramme (2014) proposed both an MILP formulation and GA with a non-greedy SGS for the problem without a minimum block length. Most recently, Tritschler et al. (2017) developed a hybrid metaheuristic for the full FRCPSP with continuous resources. Their approach is based on a non-greedy parallel SGS embedded in a GA, with an additional variable neighbourhood search improvement step.

2.4 The MRCPSP

The final extension of the classic RCPSP that we cover in this chapter on deterministic project scheduling is the multi-mode resource-constrained project scheduling problem (MRCPSP). Of all the extensions and variants of the classic RCPSP, the MRCPSP has been the subject of the most number of publications, especially in the domain of metaheuristics and GAs. However, since the MRCPSP is only considered in the context of robust scheduling in Chapter 6, here we just state the nominal version of the problem, and defer a review of the robust MRCPSP to Chapter 6. For a survey of the various formulations and solution approaches for deterministic MRCPSP, see Węglarz et al. (2011), or one of the books mentioned in the introduction of this chapter.

Introduced by Talbot (1982), the MRCPSP generalises the RCPSP to include multiple options for the processing of each activity, allowing the modelling of situations in which there is more than one way of executing project activities, with each option having its own duration and resource requirements. This approach is frequently used to model trade-offs between the duration and resource consumption of activities. Furthermore, the MRCPSP allows for the inclusion of *non-renewable* resources, i.e. resources with an overall availability for the entire project horizon. These features introduce a significant degree of flexibility for modelling and solving many real-world scheduling problems. Subject to all the same precedence and resource constraints of the standard RCPSP, a solution to the MRCPSP must determine the start time and processing mode for each project activity with the objective of minimising the project makespan. An example MRCPSP instance and its corresponding optimal solution are given in Figures 2.9 and 2.10, respectively.

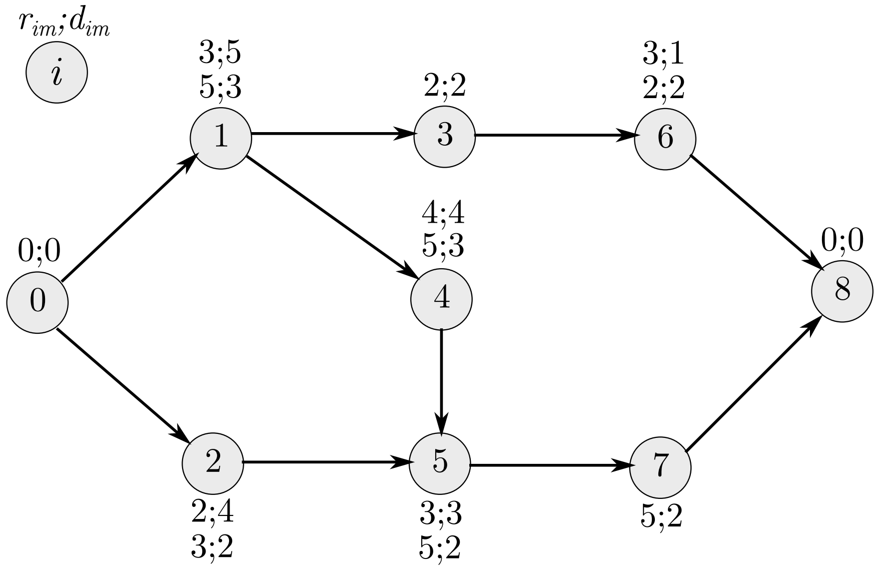


Figure 2.9: An example MRCPSP involving seven non-dummy activities and one renewable resource with $R_1 = 6$. The first modes for each activity corresponds to the data for the RCPSP instance given in Figure 2.1.

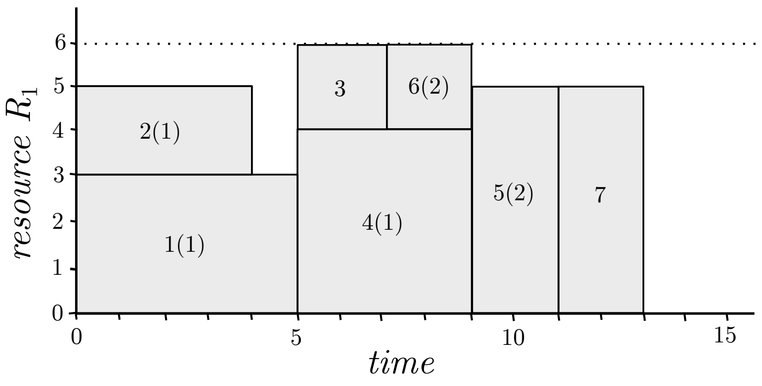


Figure 2.10: An optimal solution to the MRCPSP instance given in Figure 2.9. The activity mode choices are written in brackets.

Although the MRCPSP shares similarities with the FRCPSP, since the MRCPSP only considers a discrete number of activity processing modes with constant resource requirements across the periods of its durations, the MRCPSP does not offer the complete flexibility in the choice of resource profiles that is afforded by the FRCPSP.

Chapter 3

Robust Optimisation

In preparation for the main contributions of this thesis, this chapter aims to serve as a primer on the robust optimisation framework, with a particular focus on its application to combinatorial optimisation.

3.1 Background and motivation

Almost all real-world optimisation problems are subject to some degree of uncertainty. As mentioned in the introduction, given that the quantity and condition of the waste in some of Sellafield's oldest facilities can only be estimated, the decommissioning of Sellafield is certainly no exception to data uncertainty. Although by no means the first to do so, Ben-Tal and Nemirovski (2000) convincingly motivated the need for methods that are specifically designed to account for uncertain data when they demonstrated how deterministic solutions to uncertain optimisation problems can become catastrophically infeasible for even the smallest perturbations to the input data.

Sensitivity analysis and stochastic optimisation are two traditional approaches that are often used in an attempt to account for uncertain data, however both these approaches have major drawbacks. Sensitivity analysis is a tool for assessing the response of the output solution to changes in the input data, rather than a method for constructing a solution that is robust with respect to the input data. Furthermore, joint sensitivity analysis of multiple parameters quickly becomes impractical when the number of uncertain parameters begins to grow. Stochastic optimisation on the other hand is indeed an approach that considers the problem uncertainty in its solution. However, it makes the assumption that the uncertain data follows a known probability distribution, and most commonly formulates a problem which aims to optimise the expected long-term performance of the solution, subject to constraints that must be satisfied with a certain probability. Despite its successful application of this approach to many problems, stochastic optimisation does suffer some significant disadvantages. Firstly, accurate probabilistic knowledge of the uncertain data is very rare for real-world problems. Moreover, even if the distribution of the uncertain data is known, stochastic optimisation is typically very computationally intensive, and often results in problem instances that are too large to solve in practice.

Robust optimisation presents a different approach to accounting for uncertainty. In particular, robust optimisation makes no assumptions about the probability distributions of the inputs, but instead assumes that the uncertain data can lie anywhere inside a so-called *uncertainty set*. The problem then becomes to find solutions that are feasible for all the inputs in the specified uncertainty set, whilst optimising for some measure of the worst-case performance. As well as circumventing the need for detailed probabilistic information about the input data, robust optimisation tends to produce computationally tractable problems in many more cases than stochastic optimisation. Additionally, the worst-case performance considered by robust optimisation is, in and of itself, a valuable objective that is relevant in many contexts, particularly when a solution can only ever be evaluated once, as is frequently the case for engineering and construction projects, e.g. bridges, flood barriers, decommissioning schedules etc.

Although the first publication relating to robust optimisation dates back to the 1970s (Soyster, 1973), the field has mostly been developed over the previous 25 years following the introduction of new approaches to control the risk appetite of the robust solution, and the generalisation of robust optimisation to a range of classes of optimisation problems. Notable works that led the way to robust optimisation becoming a thriving field of research include Ben-Tal and Nemirovski (1998, 1999, 2000); El Ghaoui and Lebret (1997); El Ghaoui et al. (1998); Kouvelis and Yu (1997); Bertsimas and Sim (2004). For an extensive survey of robust optimisation, the reader is referred to Ben-Tal et al. (2009). A list of more concise survey papers include Bertsimas et al. (2011), Gabrel et al. (2014), Gorissen et al. (2015) and Goerigk and Schöbel (2016).

In the two sections that follow, the main concepts with respect to single-stage and multi-stage robust optimisation are outlined, with particular attention paid to its application to combinatorial optimisation problems. Reviews of the existing literature specifically relating to robust optimisation for project scheduling and single machine scheduling are deferred to the relevant chapters later in the thesis.

3.2 Single-stage robust optimisation

Also known as min-max, strict, absolute or static robustness, this is the classic approach to robust optimisation first developed by Soyster (1973) and refined by the works of Ben-Tal, Nemirovski, El Ghaoui and their co-authors. In this case, the decision-maker must determine a complete solution under the problem uncertainty.

Consider the following general nominal optimisation problem:

$$\begin{aligned} \min \ & f(x) \\ \text{s.t. } & g(x) \leq 0 \\ & x \in \mathcal{X}, \end{aligned} \tag{3.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ defines the objective function, $g : \mathbb{R}^n \to \mathbb{R}^m$ define a set of m constraints, and $\mathcal{X} \subseteq \mathbb{R}^n$ is some general set of variables. Denoting the uncertainty set of possible scenarios as $\mathcal{U} \subseteq \mathbb{R}^\ell$, the robust version of this optimisation problem can be written as

$$\begin{aligned} \min \max_{\xi \in \mathcal{U}} \ & f(x, \xi) \\ \text{s.t. } & g(x, \xi) \leq 0 \quad \forall \xi \in \mathcal{U} \\ & x \in \mathcal{X}, \end{aligned} \tag{3.2}$$

where $f(\cdot, \xi) : \mathbb{R}^n \to \mathbb{R}$ and $g(\cdot, \xi) : \mathbb{R}^n \to \mathbb{R}^m$ for a fixed scenario $\xi \in \mathbb{R}^\ell$. This problem aims to find a solution with the best performance in the worst-case scenario, such that the solution is feasible in all possible scenarios $\xi \in \mathcal{U}$. By replacing the objective of (3.2) by a variable $t \in \mathbb{R}$ and adding the constraints

$$t \geq f(x, \xi) \quad \forall \xi \in \mathcal{U},$$

without loss of generality, it can be assumed that the problem uncertainty resides entirely in the constraint coefficients (see Gorissen et al. (2015) for details). However, given our interest in problems with uncertain costs, for clarity of exposition, we continue to write the uncertainty in both the objective function and constraints as is the case in (3.2).

Although the approach defined in (3.2) finds a solution with the best performance in the worst-case scenario, it fails to consider that this solution may perform poorly in non-worst-case scenarios. When the problem uncertainty affects the objective function, regret is commonly used to define an alternative measure of robustness that attempts to account for this, thereby reducing the conservatism of the robust formulation. Furthermore, regret can also serve as a useful indication of how much the performance of a solution could be improved if the problem uncertainty were able to be removed. Regret robustness (also known as *absolute regret* or *absolute deviation*) replaces the objective function in (3.2) with

$$f(x, \xi) - f^*(\xi),$$

where $f^*(\xi)$ is defined to be the best possible objective value that can be achieved in scenario $\xi \in \mathcal{U}$. Another closely related measure of robustness is relative regret, which uses the objective function

$$\frac{f(x, \xi) - f^*(\xi)}{f^*(\xi)},$$

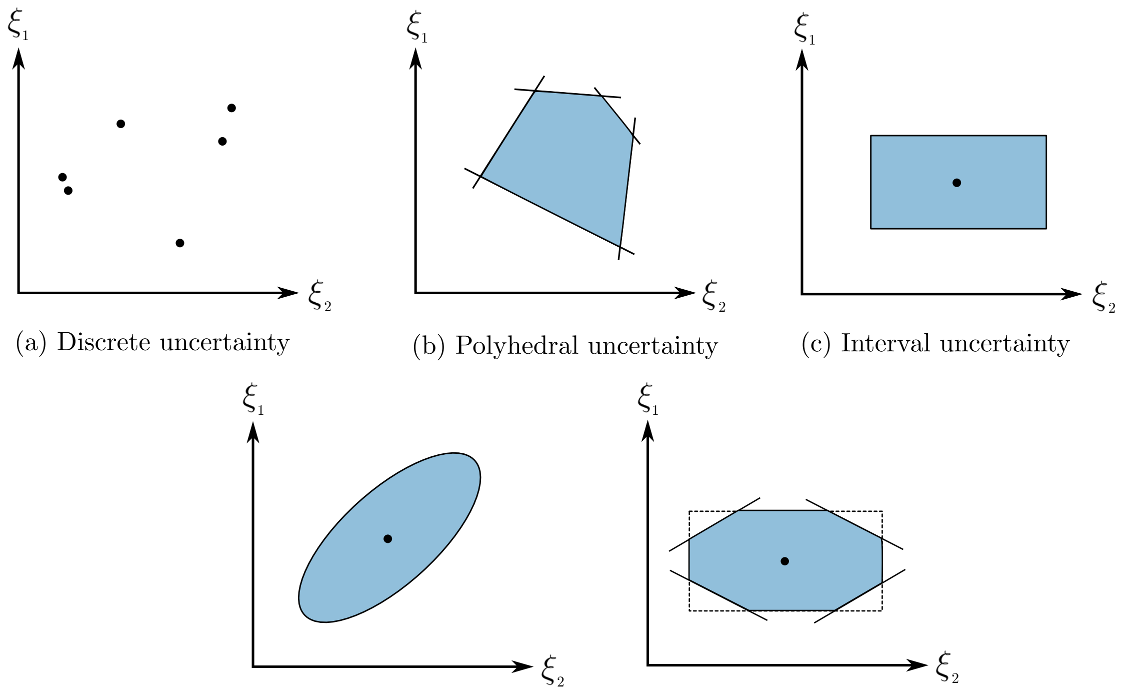
that is, the absolute regret normalised by the best objective value in scenario ξ. For a survey on regret and relative regret robustness in the context of discrete optimisation, see Kouvelis and Yu (1997) and Aissi et al. (2009).

3.2.1 Common uncertainty sets

The conservatism of the robust formulation (3.2), as well as its computational tractability, is largely influenced by the choice of the uncertainty set $\mathcal{U}$. For this reason, considerable attention has been given to the development and analysis of different types of uncertainty sets for many different problem classes. We examine the most common of these here.

Discrete uncertainty. Discrete uncertainty sets are defined simply by a finite set of possible scenarios $\mathcal{U} = \{\xi_1, \ldots, \xi_k\}$. Although discrete uncertainty sets are conceptually very simple and are an intuitive way to define uncertainty in many problems, in practice they almost always lead to intractable robust problems. For many well-known combinatorial optimisation problems that can be solved in polynomial time, their robust counterparts are NP-hard even in the case of just two scenarios. This is the case for all three measures of robustness mentioned above for the robust assignment problem, the minimum spanning tree problem and the robust shortest path problem, as well the single machine scheduling problem that we consider in chapters 7 and 8, (see Kasperski and Zieliński (2016a) and Aissi et al. (2009) for full references of these results in addition to a number of others).

Polyhedral uncertainty. Polyhedral uncertainty arises from the consideration of convex combinations of discrete scenarios and can be written in its general form as $\mathcal{U} = \left\{\xi \in \mathbb{R}^\ell : A\xi \leq b\right\}$, where $A \in \mathbb{R}^{p\times\ell}$ and $b \in \mathbb{R}^p$. For min-max robust optimisation, given that the optimal solution can be found by considering one of the extreme vertices of the uncertainty set, it can be shown that replacing



(a) Discrete uncertainty (b) Polyhedral uncertainty (c) Interval uncertainty

(d) Ellipoisal uncertainty (e) Budgeted uncertainty

Figure 3.1: Common types of uncertainty set.

a discrete uncertainty set by its convex hull does not fundamentally change the complexity of the problem. Hence, the (mostly negative) complexity results that apply to problems under discrete uncertainty also hold in the case of general polyhedral uncertainty. It is worth noting that comment does not extend in general to the two-stage problems that are the topic of the following section. Despite its general complexity, there do exist a couple of important special cases of polyhedral uncertainty for which analysis is more amenable, specifically interval uncertainty and budgeted uncertainty.

Interval uncertainty. Another natural choice of uncertainty set is the interval, or box, uncertainty set defined by the Cartesian product of intervals

$$\mathcal{U} = \{\xi \in [\underline{\xi}_1, \overline{\xi}_1] \times \cdots \times [\underline{\xi}_\ell, \overline{\xi}_\ell]\},$$

where $\underline{\xi}_i$ and $\overline{\xi}_i$ define lower an upper bounds on the range of interest of variable ξ_i. In the case of min-max robustness, supposing that all variables are positive, the worst-case scenario over $\mathcal{U}$ is simply given by the worst-case value $\overline{\xi}_i$ for each $i = 1, \ldots, \ell$. The robust problem can therefore be solved just by solving the underlying nominal problem for this worst-case scenario, and as a result the robust problem is just as simple as the nominal problem. For regret robustness however, it is not so straightforward to compute the worst-case scenario. Although the worst-case scenario is known to lie at one of the extreme points of $\mathcal{U}$ (Averbakh and Lebedev, 2004), since there are 2^ℓ such points, to enumerate them all is intractable in general. As a result, most regret-based problems of interest are NP-hard under interval uncertainty, including those that are mentioned above in the context of discrete uncertainty (again, see Kasperski and Zieliński (2016a) and Aissi et al. (2009) for the detailed references of these results). A valid criticism of the use of interval uncertainty is its strong pessimism that results from it covering scenarios in which every single uncertain parameter attains its extreme value simultaneously. With this in mind, Ben-Tal and Nemirovski (1998, 1999, 2000), El Ghaoui and Lebret (1997) and El Ghaoui et al. (1998) developed the use of ellipsoidal uncertainty sets.

Ellipsoidal uncertainty. A general ellipsoidal uncertainty set can be written in the form

$$\mathcal{U} = \left\{ \xi \in \mathbb{R}^\ell : (\xi - \hat{\xi})^\intercal \Sigma^{-1} (\xi - \hat{\xi}) \leq r^2 \right\},$$

where $\hat{\xi} \in \mathbb{R}^\ell$ defines the centre of the ellipsoid, and $\Sigma \in \mathbb{R}^{\ell \times \ell}$ is positive semi-definite. Ellipsoidal uncertainty aims to reduce the conservatism of interval uncertainty by excluding the most extreme scenarios. The size of the ellipsoid and therefore the conservatism of the resulting robust formulation can be controlled by the parameter r. Furthermore, ellipsoidal uncertainty sets allow the modelling of normally distributed uncertainty with mean $\hat{\xi}$ and covariance matrix Σ. The robust formulations that arise from the use of ellipsoidal uncertainty however are conic optimisation problems, and therefore tend to be intractable for discrete optimisation problems. However, as pointed out by Buchheim and Kurtz (2018), the complexity of robust combinatorial optimisation problems under the special case of uncorrelated ellipsoidal uncertainty (where Σ is a diagonal matrix, resulting in an ellipse that is parallel to the axes) is less known. Some complexity results for a handful of problems involving regret robustness are presented by Chassein and Goerigk (2017).

Budgeted uncertainty. Bertsimas and Sim (2004) proposed an alternative approach to reducing the so-called 'price of robustness' that involves restricting the number of parameters that can achieve their worst-case values simultaneously. We refer to their approach as *budgeted uncertainty*, however it is also known as *cardinality-constrained uncertainty* or Γ*-robustness*. Specifically, Bertsimas and Sim (2004) propose the use of the following uncertainty set:

$$\mathcal{U}(\Gamma) = \left\{ \xi \in \mathbb{R}^\ell : \xi_i \in [\hat{\xi}_i, \hat{\xi}_i + \delta_i \bar{\xi}_i],\ 0 \leq \delta_i \leq 1,\ i = 1, \dots, \ell,\ \sum_{i=1}^{\ell} \delta_i \leq \Gamma \right\}.$$

When $\Gamma = 0$, each uncertain variable takes its nominal value and the resulting robust counterpart reduces to the deterministic version of the problem. In the other extreme, when $\Gamma = \ell$, all the variables can take their worst-case values and the budgeted uncertainty set becomes equivalent to interval uncertainty. Hence the parameter Γ controls conservatism of the uncertainty set by determining the extent to which the outer regions of the interval uncertainty set are cut off. As well as being intuitive to define and use, one of the major advantages of this approach is that the resulting robust formulation shares the same computational complexity as the underlying nominal problem. As a result, this approach can be directly applied to discrete optimisation problems, as examined by Bertsimas and Sim (2003). Subsequently, budgeted uncertainty has become the most widely used and studied uncertainty set for a range of combinatorial optimisation problems.

3.3 Two-stage Robust Optimisation

Ben-Tal et al. (2004) were the first to extend the robust optimisation framework to a two-stage setting. Following this, much of the research relating to robust optimisation, including the primary contributions of this thesis, has been concerned with the analysis and application of two-stage approaches for many different problem types. This section summarises the main approaches in the literature to two-stage robust optimisation. For a discussion on the extension of the robust optimisation to more than two stages, see Delage and Iancu (2015).

3.3.1 Adjustable robustness

First introduced by Ben-Tal et al. (2004) for linear optimisation, adjustable robust optimisation (also known simply as two-stage robust optimisation) makes the assumption that the problem decision variables can be split into two categories: 1. variables that must be determined under the problem uncertainty, i.e. *here-and-now variables*, and 2. variables that can be decided once the actual scenario $\xi \in \mathcal{U}$ becomes known, i.e. *wait-and-see variables*. By allowing a subset of the decision variables to be determined after the realisation of the uncertain data, a greater degree of flexibility is offered compared to classic one-stage robust optimisation, resulting in a reduction in conservatism and an improvement in the objective value of the robust solution. Furthermore, this two-stage approach models many real-world decision processes quite naturally, including the scheduling process considered in Chapters 5 and 6 of this thesis. Denoting the feasible sets of the first and second-stage variables by $\mathcal{X}^1 \subseteq \mathbb{R}^{n_1}$ and $\mathcal{X}^2 \subseteq \mathbb{R}^{n_2}$ respectively, where $n_1 + n_2 = n$, the adjustable robust optimisation problem can be written as

$$\begin{aligned} &\min_{x \in \mathcal{X}^1} \max_{\xi \in \mathcal{U}} \min_{y(\xi) \in \mathcal{X}^2} f(x, y(\xi), \xi) \\ &\qquad\qquad \text{s.t. } g(x, y(\xi), \xi) \leq 0 \quad \forall \xi \in \mathcal{U}. \end{aligned} \tag{3.3}$$

The first-stage variables $x \in \mathcal{X}^1$ must be chosen such that for any possible scenario $\xi \in \mathcal{U}$ there exists second-stage variables $y \in \mathcal{X}^2$ such that the feasibility constraints can be satisfied, whilst minimising the objective function.

Observe that the single-stage robust optimisation problem (3.2) is a special case of this problem obtained simply by setting $n_2 = 0$ and forcing all the second-stage variables to be specified under the problem uncertainty in the first stage. Clearly therefore, the adjustable robust problem is at least as hard as the single-stage problem. Considering problems with uncertainty only in the objective function, if the sets $\mathcal{U}$, $\mathcal{X}^1$ and $\mathcal{X}^2$ are all convex, then by making use of the minimax theorem it can be shown that the adjustable robust problem is equivalent to the min-max problem. For problems that involve uncertainty in the constraints however, in general, (3.3) is NP-hard, even if the corresponding min-max problem is polynomially solvable. As a result, much of the research regarding adjustable robust optimisation has been focused on the development of approximations for solving this problem.

Supposing that the second-stage recourse variables are continuous, Ben-Tal et al. (2004) proposed an approximation to (3.3) based on assuming that the second-stage variables are affine functions of the problem uncertainty, i.e. $y = y_0 + Q\xi$, where $y_0 \in \mathbb{R}^{n_2}$ and $Q \in \mathbb{R}^{n_2 \times \ell}$. In this case, the second-stage decision variables y are replaced by y_0 and Q and must be determined in the first-stage subject to the problem uncertainty. Supposing fixed recourse, it is shown that this problem has the same complexity as the single-stage min-max robust problem. Furthermore, it has been shown that for many problems, these affine decision rules obtain optimal, or near optimal results, encouraging the use of this approximation to solve a wide range of problems. We note that there exist many alternative approaches for the tractable approximation of (3.3), as well as a number of exact decomposition-based approaches, including, most notably, a row-and-column generation approach for the case of polyhedral uncertainty developed by Zeng and Zhao (2013).

Adjustable robust optimisation with integer recourse is of particular interest with regards to combinatorial optimisation, however this setting has received considerably less attention in the literature than the case of continuous recourse mentioned above. Kasperski and Zieliński (2011), Kasperski and Zieliński (2017), Chassein et al. (2018), Goerigk et al. (2021a) and Goerigk et al. (2022a) all consider this model for a range of combinatorial optimisation problems including selection, spanning tree and shortest path problems and show that for discrete or general convex uncertainty sets the resulting adjustable robust problem is hard. Under interval and continuous budgeted uncertainty, a number of more positive complexity results are derived. Chapters 5 and 6 of this thesis consider adjustable robustness for the RCPSP and MRCPSP with uncertain activity durations.

For a review of adjustable robust optimisation and its various approximations and solution approaches, see Yanıkoğlu et al. (2019).

3.3.2 Recoverable robustness

Developed by Liebchen et al. (2009) for the problem of timetabling trains, recoverable robustness is an alternative approach to two-stage robust optimisation. Instead of determining a partial solution under the problem uncertainty and then completing the solution once the uncertain data becomes known, recoverable robustness constructs a complete solution under the problem uncertainty, whilst accounting for a set of recovery actions that can be applied to the solution once the uncertainty has been revealed.

Letting $\mathcal{A}$ denote the set of recovery actions, the general recoverable robust optimisation problem can be written as

$$\begin{aligned} &\min_{x \in \mathcal{X}} f(x) \\ &\text{s.t. } g(A(x,\xi),\xi) \leq 0 \quad \forall \xi \in \mathcal{U} \\ &\qquad A \in \mathcal{A}. \end{aligned} \tag{3.4}$$

That is, a solution $x \in \mathcal{X}$ and recovery action $A \in \mathcal{A}$ must be chosen such that x minimises f and is feasible following its recovery after the realisation of the actual scenario. In the case of objective uncertainty, the recoverable robust approach can be reinterpreted as having the aim of finding a feasible solution $x \in \mathcal{X}$ with the best possible performance in the nominal scenario, subject to being recovered to optimality once the uncertainty has been revealed. This model can be extended to also include the cost of the recovery of the first-stage solution. For any particular problem, clearly, the complexity of the recoverable robust version very much depends on the specification of the set of recovery actions $\mathcal{A}$, as well as on the choice of uncertainty set $\mathcal{U}$.

Given its generality, the recoverable approach to robust optimisation has seen numerous applications in recent years to a range of problems including the knapsack problem (Büsing et al., 2011), shortest path problem (Büsing, 2012), travelling salesman problem (Chassein and Goerigk, 2016), selection problem Chassein et al. (2018); Kasperski and Zieliński (2017); Goerigk et al. (2022b) and assignment problem (Fischer et al., 2020), among others. In chapters 7 and 8, recoverable robustness is applied to the single machine scheduling problem for the first time.

3.3.3 K-adaptability

Bertsimas and Caramanis (2010) introduced an approach to approximating the adjustable robust problem (3.3) based on the idea of determining K solutions under the problem uncertainty in the first stage, before selecting the best of these K options in the second stage, once the uncertain scenario becomes known. This so-called K-*adaptability* problem can be written as

$$\begin{aligned} \min_{\substack{x\in\mathcal{X}^1\\ y^1,\dots,y^K\in\mathcal{X}^2}} \max_{\xi\in\mathcal{U}} \min_{i=1,\dots,K} &\ f(x,y^i,\xi) \\ \text{s.t. } &\ g(x,y^i,\xi)\le 0 \quad \forall \xi\in\mathcal{U}. \end{aligned} \tag{3.5}$$

This approach of preparing K solutions subject to uncertainty is particularly well suited to model many real-world decision problems and its application has been widespread. Example include hub location (Alumur et al., 2012) and parcel deliveries (Eufinger et al., 2020).

Hanasusanto et al. (2015) apply the K-adaptability approach to two-stage robust binary optimisation, and show that if the problem uncertainty resides only in the objective function, then an optimal solution can be found by using $K = n + 1$ second-stage solutions. This result is used to derive one of the formulations proposed for the recoverable robust single-machine scheduling problem considered in Chapter 7 of this thesis. This result does not extend to the general two-stage robust binary problem where uncertainty is also present in the constraints, in which case the problem is NP-hard. Recently Subramanyam et al. (2020) have extended K-adaptability to mixed-integer programming and propose a branch-and-bound scheme to solve the resulting problem.

As a special case of K-adaptability, Buchheim and Kurtz (2017) study two-stage robust combinatorial optimisation problems of the form

$$\min_{x^1,\dots,x^K \in \mathcal{X}} \max_{\xi \in \mathcal{U}} \min_{i=1,\dots,K} f(x^i, \xi) \tag{3.6}$$

where there are no first-stage variables and all the problem uncertainty is contained within the objective function. They refer to this approach as *min-max-min robustness*. Buchheim and Kurtz (2017) show that when $K \geq n + 1$, for convex uncertainty sets, this problem has the same complexity as the underlying deterministic problem. As shown by Buchheim and Kurtz (2016), this result does not extend to discrete uncertainty, because, unlike the case for min-max robustness, replacing a discrete uncertainty set $\mathcal{U}$ by its convex hull in (3.6) fundamentally changes the problem. Whilst proving that the min-max-min problem is NP-hard for discrete uncertainty, Buchheim and Kurtz (2016) also propose a pseudopolynomial-time algorithm to reduce (3.6) to its min-max version, implying that pseudopolynomial-time solution algorithms to (3.6) exist for a number of combinatorial problems. In a series of recent papers Chassein et al. (2019); Goerigk et al. (2020); Chassein and Goerigk (2021), the min-max-min problem has been studied under both discrete and continuous budgeted uncertainty sets. For the continuous set, positive complexity results are proved for a range of optimisation problems, however, unsurprisingly these do not extend to discrete budgeted uncertainty.

Chapter 4

The Generalised Flexible Resource-Constrained Project Scheduling Problem

4.1 Introduction

The decommissioning of the Sellafield nuclear site in North West England is one of the largest and most complex ongoing engineering projects in Europe. It is expected to take in excess of 100 years to complete and cost a total of over £90 billion (NDA, 2019). Given its scale and complexity, it is crucial that this project is choreographed according to a carefully designed master schedule that fully accounts for the network of logical precedence relationships between the decommissioning activities, as well as the limited availability of the project resources. This chapter introduces and solves a new project scheduling model designed to schedule this project and others like it.

The resource-constrained project scheduling problem (RCPSP) has been thoroughly studied since the introduction of a first model by Pritsker et al. (1969). The RCPSP consists of scheduling a set of activities, subject to resource and precedence constraints, in order to minimise the overall project duration, known as the makespan. Although a very general scheduling model, the applicability of the classical RCPSP to many real-world problems, including the Sellafield nuclear decommissioning project, is limited by the following two assumptions: 1. only finish-to-start, zero time-lag precedence relationships exist between activities, and 2. the resource requirements of the activities are fixed and constant throughout their duration. This chapter introduces and solves an extension to the classical RCPSP that allows for both of these assumptions to be relaxed, enabling the Sellafield nuclear decommissioning project and other projects with similar features, to be modelled and scheduled.

Extensions to the RCPSP that address one of these two limiting assumptions are well-studied. The addition of generalised precedence relationships to the RCPSP addresses the first of these. Although a number of different names are used to refer to this extenstion of the RCPSP, here we refer to it as the generalised resource-constrained project scheduling problem (GRCPSP). More recently, attention has turned to addressing the second limiting assumption of the classical RCPSP, with the introduction of the resource-constrained project scheduling problem with flexible resource-profiles (FRCPSP) by Naber and Kolisch (2014). In this problem, it is assumed only that the total amount of resource required to complete each activity is known, and that, as well as the start time, the resource allocation for each activity throughout its duration must be determined, subject to a set of constraints on that allocation.

To the best of our knowledge, up until now, no model has been introduced that simultaneously relaxes both of these assumptions. By introducing the generalised flexible resource-constrained project scheduling problem (GFRCPSP) in this chapter, we combine the two extensions mentioned above into a single model capable of accurately capturing the features of a much wider range of real-world projects than the standard RCPSP.

The GFRCPSP is a very challenging problem. In particular, due to the introduction of generalised precedence constraints, the decision problem of determining whether or not a feasible solution exists for a given GFRCPSP instance is an NP-complete problem (Bartusch et al., 1988), and in practice, for many problem instances, just finding a feasible solution is the primary difficulty. Hence, a solution approach must be designed to prioritise the finding of feasible solutions, before it then works to improve upon these feasible solutions. Therefore, as well as introducing a mixed-integer programming (MIP) formulation, we propose a genetic algorithm for finding good solutions to production-sized instances. A computational study demonstrates the strong performance of the proposed metaheuristic algorithm when compared with solving the proposed MIP using a state-of-the-art solver, as well as four additional benchmarking heuristics. The applicability of this newly developed model and the proposed metaheuristic is further demonstrated by the modelling and scheduling of a decommissioning project from the Sellafield site.

The remainder of this chapter is organised as follows: Section 4.2 formally introduces the GFRCPSP. Section 4.3 presents an MIP formulation for the GFRCPSP, before Section 4.4 details a genetic algorithm-based solution approach. Section

4.5 presents and compares computational results from the proposed metaheuristic algorithm, the MIP formulation and four additional benchmarking heuristics. Section 4.6 applies the methods proposed in the earlier sections to the scheduling of a Sellafield nuclear decommissioning project. Finally, concluding remarks are given in Section 4.7.

4.2 Problem description

A project consists of a set of non-preemptive activities $V = \{0, 1, \ldots, n, n+1\}$, where activities 0 and $n+1$ are dummy project-start and project-end activities with duration 0 and no resource requirements. We let $N = \{1, \ldots, n\}$ denote the set of non-dummy activities. The start time and resource profile of each activity must be determined over a planning horizon of discrete time periods $t \in T$, subject to a set of precedence and resource constraints. We first detail the project precedence constraints.

4.2.1 Precedence constraints

There are four possible types of generalised precedence relationship between two activities: start-to-start, start-to-finish, finish-to-start and finish-to-finish, and each generalised precedence relationship has an associated minimal or maximal time-lag. For example, consider a finish-to-start type relationship between activities i and j. If this relationship has a minimal time-lag of length $a \geq 0$, the resulting constraint requires that $s_j \geq f_i + a$, where s_j and f_i are the start and finish times of activities j and i, respectively. That is, j cannot start until a time periods after the finish of i. If there is also a maximal time-lag of length

$b \geq a$ between the finish of i and the start of j, the resulting constraint requires that $s_j \leq f_i + b$, i.e. j must have started by b time periods after the finish of i. These two precedence relationships combine to create a feasible time-window $\{f_i + a, \ldots, f_i + b\}$ in which activity j must start, relative to the finish time of activity i.

For the GRCPSP, since the duration of each activity is known, the four types of generalised precedence constraints are in fact equivalent and any relationship type can be transformed into any other (see Bartusch et al. (1988)). A standard pre-processing step when solving the GRCPSP is therefore to transform all the project precedence relationships into a single type, typically start-to-start. In our setting however, to avoid time-lags depending on the variable activity durations, these transformations are not applied and the different precedence relationships types are considered separately.

It is important to recognise that maximal time-lags can be rewritten as negative minimal time-lags going in the opposite direction. For example, again consider a maximal time-lag of length b from f_i to s_j given by $s_j \leq f_i + b$. This maximal time-lag can be rewritten as $f_i \geq s_j - b$; that is, as a minimal time-lag of length $-b$ from s_j to f_i. Having converted all maximal time-lags into minimal time-lags, we can represent a finish-to-start precedence relationship of length a between i and j as a tuple (i, j, a), where a may be either positive or negative. The other types of generalised precedence relationship can also be represented in this way, resulting in a set for each type of relationship, which we denote by E_{SS}, E_{SF}, E_{FS} and E_{FF}.

Additionally, once all maximal time-lags have been converted into minimal time-lags, the project precedence constraints can be represented on a network.

An example project network for a GFRCPSP instance involving five non-dummy activities is shown in Figure 4.1. Note the occurrence of a cycle involving activities 2, 3 and 4. Although not the case with this example, if a cycle of positive length exists in the network, it can immediately be determined that the project is infeasible with respect to the precedence constraints.

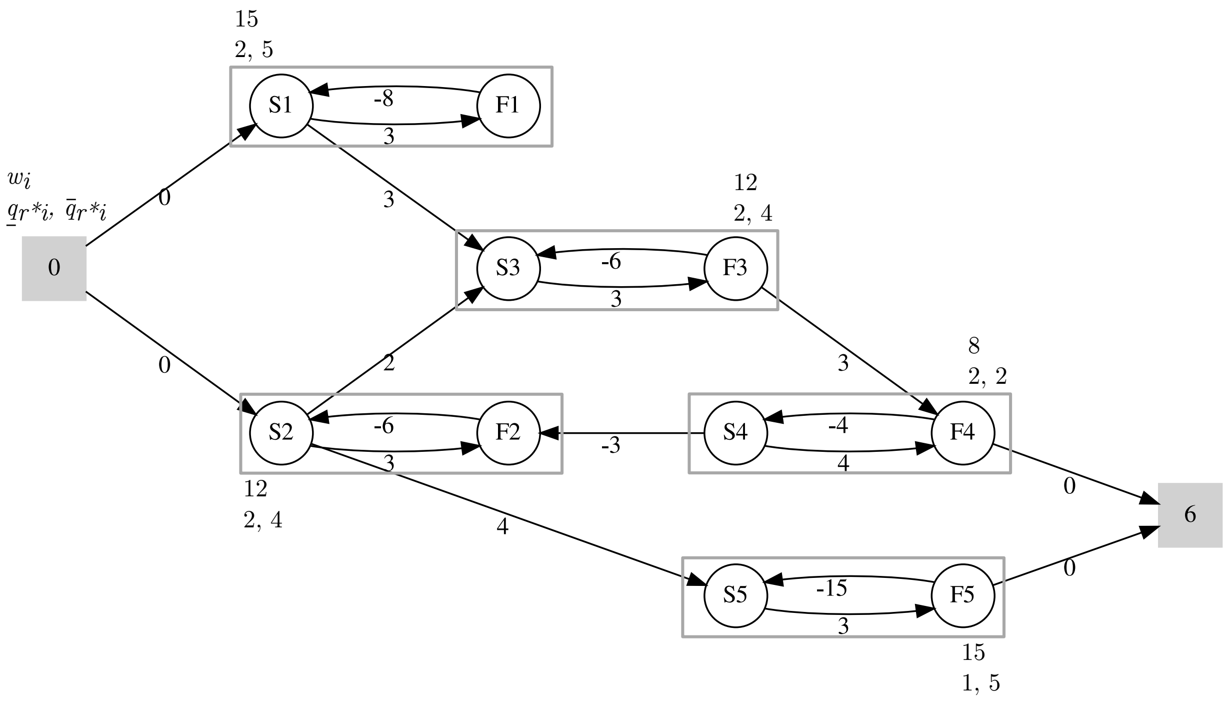


Figure 4.1: An example GFRCPSP instance with five non-dummy activities and a single resource r^*, with availability $R_{r^*}^{\max} = 6$. For each activity $i \in V$, the total principal resource requirement w_i, and upper and lower bounds on its per-period allocation $\underline{q}_{r^*i}$, $\overline{q}_{r^*i}$, are shown. Minimal time-lags are shown next to each arc in the network.

Given a set of generalised precedence constraints, earliest and latest start and finish times of each activity $i \in N$ can be calculated using the pre-processing step outlined in Appendix A. We denote these values by ES_i, LS_i, EF_i, and LF_i, respectively. The sets of time periods in which an activity $i \in V$ must start and finish are given by $ST_i = \{ES_i, \dots, LS_i\}$ and $FT_i = \{EF_i, \dots, LF_i\}$, respectively. A feasible time-window for the processing of activity i is given by $T_i = \{ES_i, \dots, LF_i\}$. In addition to these time-windows, an upper bound on the minimum project makespan, $T^{\max}$, is also computed in this pre-processing step.

4.2.2 Resource constraints

We now detail the resource constraints. We follow the definition of the resource constraints of the FRCPSP as presented in Naber and Kolisch (2014) and assume that each resource $r \in R$ is renewable, continuously divisible, and has a limited availability of $R_r^{\max}$ at each time period $t \in T$. Furthermore, for each activity $i \in N$, each resource $r \in R$ is categorised into one of the following three types of resource:

1. **The principal resource** r^* of activity i is the main resource used by that activity. The amount of principal resource allocated to i entirely determines the amount of dependent resource allocated to i.

2. **A dependent resource** r of activity i is a resource for which the amount allocated to i depends on the amount of principal resource allocated to i. More specifically, if q_{r^*it} is the amount of principal resource r^* allocated to activity i in time period t, $q_{rit} = \alpha_{ri} q_{r^*it} + \beta_{ri}$ gives the amount of dependent resource r allocated to i at time t, where $\alpha_{ri} = (\overline{q}_{ri} - \underline{q}_{ri})/(\overline{q}_{r^*i} - \underline{q}_{r^*i})$ and $\beta_{ri} = \underline{q}_{ri} - \underline{q}_{r^*i}\alpha_{ri}$ are the coefficient and constant of the non-decreasing linear resource function that links the dependent and principal resource allocations. $\underline{q}_{ri}$ and $\overline{q}_{ri}$ are lower and upper bounds on the allocation of resource r to activity i in any given time period. The set of dependent resources of activity

i is denoted by R_i^{dep}.

3. **An independent resource** r of activity i is a resource with allocation that is independent from the allocated quantity of any other resources. The set of independent resources of activity i is denoted by R_i^{ind}.

For each activity $i \in N$, the allocation of each resource $r \in R$ at each time period $t \in T$ must be determined. This forms the *resource profile* of activity i. The resource profile of each activity $i \in N$ is subject to the following constraints:

1. The total amount of principal resource allocated to activity i over its duration must at least satisfy a required amount, denoted by w_i.

2. There is an upper bound, $\overline{q}_{ri}$, and lower bound, $\underline{q}_{ri}$, on the amount of resource r that can be allocated to i for each period in which activity i is being processed. The upper bound on the principal resource allocation provides a lower bound on the duration of activity i, given by $\underline{d}_i = \lceil w_i/\overline{q}_{r^*i} \rceil$. Similarly, the lower bound on the principal resource allocation provides an upper bound, given by $\overline{d}_i = \lceil w_i/\underline{q}_{r^*i} \rceil$. Note that since we allow the total resource requirement of each activity to be exceeded, this upper bound is not strictly necessary and may be chosen differently or omitted entirely. The minimum and maximum duration of each activity $i \in N$ can be represented as a minimal and a maximal time-lag from s_i to f_i, respectively. Recall that the maximal time-lag representing the maximum duration of i can be converted into a negative minimal time-lag of length $-\overline{d}_i$ from f_i to s_i.

3. There is a minimum number of consecutive time periods for which the resource allocation to an activity must be constant. This is known as the

minimum block length and is denoted by $l^{\min}$.

The GFRCPSP problem consists of finding a start time and resource profile for each activity $i \in N$ that is feasible with respect to the precedence and resource constraints outlined above, to minimise the project makespan. Figure 4.2 shows an optimal solution to the GFRCPSP represented by the network in Figure 4.1. Table 4.1 provides a summary of the notation used throughout this chapter.

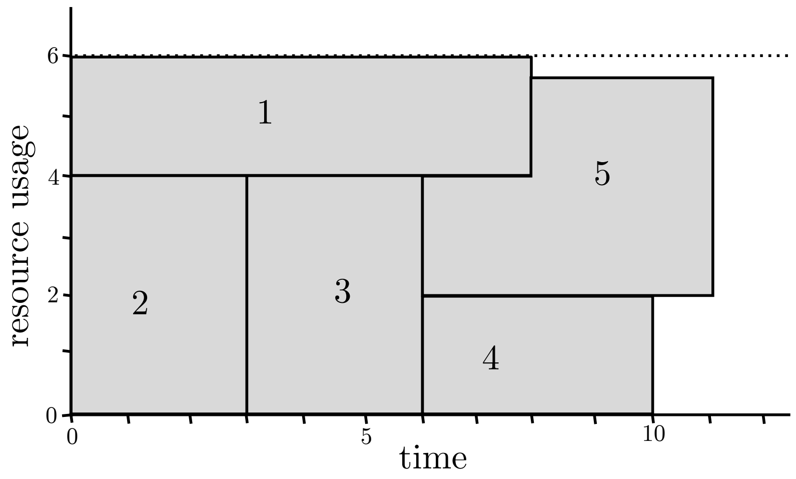


Figure 4.2: An optimal solution to the GFRCPSP instance shown in Figure 4.1.

4.3 Mixed-integer programming formulation

Of the four MIP models for the FRCPSP introduced by Naber and Kolisch (2014), the so-called *variable-intensity-based* model (Model FP-DT3), based on the RCPSP model of Bianco and Caramia (2013), was shown to be the strongest. This section details an extension to this model to include generalised precedence constraints.

This model uses 'intensity' variables ν_{it} to represent the proportion of the required principal resource w_i that has been allocated to activity i by time t. These intensity variables are linked to variables q_{rit} which track the allocation of

Indices	
i, j	Activities
r	Resource (of unspecified type)
t	Time period
Parameters	
ES_i, LS_i, EF_i, LF_i	Earliest and latest start and finish times of activity i
r_i^*	Principle resource of activity i
$R_r^{\max}$	Availability of resource r
$\underline{q}_{ri}$, $\overline{q}_{ri}$	Lower and upper bounds on per-period allocation of resource r to activity i
w_i	Total amount of principal resource required by activity i
$l^{\min}$	Minimum block length
α_{ri}, β_{ri}	Coefficient and constant of the linear resource function linking q_{rit} to q_{r^*it}
$\underline{d}_i$, $\overline{d}_i$	Minimum and maximum duration of activity i
$T^{\max}$	Upper bound on the minimal project makespan
Variables	
s_i, f_i	Start and finish time of activity i
q_{rit}	Amount of resource r allocated to activity i in time period t
Index sets	
$V = \{0, 1, \ldots, n, n+1\}$	Set of activities (including dummy activities)
$N = \{1, \ldots, n\}$	Set of non-dummy activities
$T = \{0, 1, \ldots, T^{\max}\}$	Project planning horizon
$T_i = \{ES_i, \ldots, LF_i\}$	Feasible processing periods of activity i
$ST_i = \{ES_i, \ldots, LS_i\}$	Feasible starting periods of activity i
$FT_i = \{EF_i, \ldots, LF_i\}$	Feasible finishing periods of activity i
R	Set of resources
R_i^{dep}, R_i^{ind}	Sets of dependent and independent resources of activity i
E_{SS}, E_{SF}, E_{FS}, E_{FF}	Set of start-to-start, start-to-finish, finish-to-start and finish-to-finish minimum time-lag precedence relations, e.g. $(i, j, a) \in E_{FS}$ represents the constraint $s_j \geq f_i + a$

Table 4.1: Summary of notation.

resource r to activity i in time period t. Variables x_{it} indicate whether activity i has started by time period t, and similarly, variables y_{it} indicate whether activity i has finished by time period t. Consequently, the processing status of activity i can be determined by $x_{it} - y_{it}$ and the start and finish times of activity i can be written as $s_i = LF_i - \sum_{t \in T_i} x_{it}$ and $f_i = LF_i - \sum_{t \in T_i} y_{it}$, respectively. Finally, in order to satisfy the minimum block length condition, variables δ_{it} track the changes in the quantity of principal resource allocated to activity i. Table 4.2 summarises the decision variables used in this formulation.

Binary variables		
x_{it}	$\begin{cases} 1, & \text{if activity } i \text{ starts at or before time } t, \\ 0, & \text{otherwise,} \end{cases}$	$\forall i \in N,\ t \in T_i$
y_{it}	$\begin{cases} 1, & \text{if activity } i \text{ ends at or before time } t, \\ 0, & \text{otherwise,} \end{cases}$	$\forall i \in N,\ t \in T_i$
δ_{it}	$\begin{cases} 1, & \text{if } q_{r^*i,t-1} \neq q_{r^*it}, \\ 0, & \text{otherwise,} \end{cases}$	$\forall i \in N,\ t \in T_i \cup \{LF_i + 1\}$
Continuous variables		
$C^{\max}$	Project makespan	
q_{rit}	Quantity of resource $r \in R$ allocated to activity $i \in N$ in time period $t \in T_i \cup \{ES_i - 1, LF_i + 1\}$	
ν_{it}	Proportion of activity $i \in N$ completed by time period $t \in T_i$	

Table 4.2: Decision variables used in formulation (4.1)-(4.29).

Using the notation and variables from Tables 4.1 and 4.2, respectively, an MIP model for the GFRCPSP can be formulated as follows:

$$\min C^{\max} \tag{4.1}$$

$$\text{s.t. } C^{\max} \geq LF_i - \sum_{t \in T_i} y_{it} + 1 \qquad \forall i \in N \tag{4.2}$$

$$q_{rit} - \overline{q}_{ri}(x_{it} - y_{it}) \leq 0 \qquad \forall r \in R,\ i \in N,\ t \in T_i \tag{4.3}$$

$$q_{rit} - \underline{q}_{ri}(x_{it} - y_{it}) \geq 0 \qquad \forall r \in R,\ i \in N,\ t \in T_i \tag{4.4}$$

$$q_{rit} \geq \alpha_{ri} q_{r^*it} + \beta_{ri}(x_{it} - y_{it}) \qquad \forall r \in R_i^{\text{dep}},\ i \in N \tag{4.5}$$

$$q_{r^*it} \geq w_i(\nu_{i,t+1} - \nu_{it}) \qquad \forall i \in N,\ t \in T_i \tag{4.6}$$

$$\sum_{\{i \in N : t \in T_i\}} q_{rit} \leq R_r^{\max} \qquad \forall r \in R,\ t \in T \tag{4.7}$$

$$\sum_{\tau=t}^{t+l^{\min}-1} \delta_{i\tau} \leq 1 \qquad \forall i \in N, t \in \{ES_i, \ldots, LF_i - l^{\min} + 2\} \tag{4.8}$$

$$q_{r^*it} - q_{r^*i,t-1} - \overline{q}_{r^*i}\delta_{it} \leq 0 \qquad \forall i \in N,\ t \in T_i \cup \{LF_i + 1\} \tag{4.9}$$

$$q_{r^*i,t-1} - q_{r^*it} - \overline{q}_{r^*i}\delta_{it} \leq 0 \qquad \forall i \in N,\ t \in T_i \cup \{LF_i + 1\} \tag{4.10}$$

$$q_{ri,ES_i-1} = q_{ri,LF_i+1} = 0 \qquad \forall r \in R,\ i \in N \tag{4.11}$$

$$\nu_{it} \leq x_{it} \qquad \forall i \in N,\ t \in ST_i \tag{4.12}$$

$$\nu_{it} \geq y_{it} \qquad \forall i \in N,\ t \in FT_i \tag{4.13}$$

$$x_{j,t+a} \leq x_{it} \qquad \forall (i,j,a) \in E_{SS},\ t \in ST_i \cup \{ES_j - a, \ldots, LS_j - a\} \tag{4.14}$$

$$y_{j,t+b} \leq x_{it} \qquad \forall (i,j,b) \in E_{SF},\ t \in ST_i \cup \{EF_j - b, \ldots, LF_j - b\} \tag{4.15}$$

$$x_{j,t+c} \leq y_{it} \qquad \forall (i,j,c) \in E_{FS},\ t \in FT_i \cup \{ES_j - c, \ldots, LS_j - c\} \tag{4.16}$$

$$y_{j,t+d} \leq y_{it} \qquad \forall (i,j,d) \in E_{FF},\ t \in FT_i \cup \{EF_j - d, \ldots, LF_j - d\} \tag{4.17}$$

$$\nu_{it} \leq \nu_{i,t+1} \qquad \forall i \in N,\ t \in T_i \tag{4.18}$$

$$x_{i,t-1} \leq x_{it} \qquad \forall i \in N,\ t \in T_i \setminus \{ES_i\} \tag{4.19}$$

$$y_{i,t-1} \leq y_{it} \qquad \forall i \in N,\ t \in T_i \setminus \{ES_i\} \quad (4.20)$$

$$\nu_{i,ES_i} = 0 \qquad \forall i \in N \quad (4.21)$$

$$\nu_{i,LF_i} = 1 \qquad \forall i \in N \quad (4.22)$$

$$x_{it} = 1 \qquad \forall i \in N,\ LS_i \leq t \leq LF_i \quad (4.23)$$

$$y_{it} = 0 \qquad \forall i \in N,\ ES_i \leq t \leq EF_i - 1 \quad (4.24)$$

$$q_{rit} \geq 0 \qquad \forall r \in R,\ i \in N,\ t \in T_i \cup \{ES_i - 1, LF_i + 1\} \quad (4.25)$$

$$0 \leq \nu_{it} \leq 1 \qquad \forall i \in N,\ t \in T_i \quad (4.26)$$

$$x_{it} \in \{0,1\} \qquad \forall i \in N,\ t \in T_i \quad (4.27)$$

$$y_{it} \in \{0,1\} \qquad \forall i \in N,\ t \in T_i \quad (4.28)$$

$$\delta_{it} \in \{0,1\} \qquad \forall i \in N,\ t \in T_i \cup \{LF_j + 1\} \quad (4.29)$$

Constraints (4.2) define the makespan to be the latest completion time of an activity in the project. Constraints (4.3) and (4.4) enforce the upper and lower bounds on the resource allocation for each activity in the time periods that it is being processed. Constraints (4.5) determine the allocation of each dependent resource according to the linear function of the allocation of principal resource. Constraints (4.6) link the resource allocation variables q_{rit} to the intensity variables ν_{it}, and constraints (4.7) ensure that the total amount of each resource used in each period does not exceed the availability. Constraints (4.8) ensure that the minimum block length is satisfied, whilst constraints (4.9) and (4.10) make sure that the δ_{it} variables track changes in the resource allocation as intended. Constraints (4.11) initialise the resource allocation variables to zero outside the feasible processing time window of each activity. Constraints (4.12) and (4.13) ensure that the processing of an activity can only occur between its start and finish times. Constraints (4.14)-(4.17) ensure that the generalised precedence constraints are satisfied. Note that these constraints are the only significant difference between the MIP formulation presented here, and the FP-DT3 formulation presented in Naber and Kolisch (2014). Constraints (4.18)-(4.20) are non-preemption constraints, and constraints (4.21)-(4.24) initialise the variables with known values. Finally, the variables are defined by constraints (4.25)-(4.29).

4.4 Metaheuristic algorithm

In this section we present a scheduling heuristic and genetic algorithm to solve the GFRCPSP. Section 4.4.1 outlines the proposed flexible schedule generation scheme, before the genetic algorithm into which this scheduling heuristic is embedded is detailed in Section 4.4.2.

4.4.1 A flexible schedule generation scheme

A new non-greedy flexible serial schedule generation scheme (FSGS) has been developed specifically for GFRCPSP. This FSGS takes an activity list and constructs a complete solution by scheduling activities one at a time in the order given by the activity list. Activities are scheduled to start as early as possible and with as much resource allocated as possible, whilst respecting enforced delays on start times, limits on resource allocations, and the project precedence constraints. These delays and resource limits enable the FSGS to perform non-greedy scheduling. The following three sections list the input parameters required by the FSGS, detail the specific steps performed by the algorithm, and provide an example of its use, respectively.

Input parameters

The input parameters required by this FSGS are the same parameters used in the FSGS developed by Tritschler et al. (2017) for the FRCPSP. These are:

1. **Activity list $\boldsymbol{\lambda}$.** A precedence feasible permutation of the project activities $i \in V$, specifying the order in which the activities are added to the schedule.

2. **Greediness parameters $\boldsymbol{\rho}$.** A list $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_n)$ of resource allocation limiting parameters for each non-dummy project activity $i \in N$. $\rho_i \in \{0, \ldots, \overline{\rho}_i\}$, where $\overline{\rho}_i = \lceil w_i / \underline{q}_{r^*i} \rceil - \underline{d}_i$. The resource allocation limit ρ_i determines the maximum principal resource allocation to activity i through the function $\overline{q}_{r^*it} = w_i / (\underline{d}_i + \rho_i)$.

3. **Delay parameters $\boldsymbol{\sigma}$.** A list $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ of start delay parameters for each non-dummy project activity $i \in N$. $\sigma_i \in \{0, \ldots, \overline{\sigma}_i\}$, where $\overline{\sigma}_i = \min(\lceil w_i / \underline{q}_{r^*i} \rceil - \underline{d}_i, LF_i - ES_i)$. Delay parameter σ_i specifies the number of time periods by which the start of activity i must be delayed beyond its earliest feasible start time.

Algorithm

The FSGS we present here extends the serial SGS for application to the GFRCPSP. Algorithm 3.1 shows the steps performed by the FSGS in scheduling a single activity $i \in N$. For the purposes of this initial exposition, it is assumed that the schedule produced by Algorithm 3.1 is time-feasible, and therefore that the unscheduling step is not required.

In Algorithm 3.1, ξ_i denotes the remaining principal resource requirement of activity i, and φ_{rt} denotes the remaining availability of resource $r \in R$ at time t. l_{it} denotes the time since the last change in resource allocation for activity i at time t. Note that, for the sake of brevity, the parameter updates that occur at the end of each **while** loop, as well as the allocation of independent resource, have been omitted from Algorithm 3.1. Note also that for each $i \in V$, l_{it} is initialised to $l^{\min}$. The steps of Algorithm 3.1 are outlined below.

Firstly, t is initialised in line 1. While either the current resource block of activity i is incomplete or its resource requirement has not yet been met (line 2), resource is allocated to activity i one period at a time. When determining the amount of principal resource to allocate to activity i at time t, two situations may occur. 1. If the current resource block has not yet satisfied the minimum block length (line 3), the current resource block is extended into the next time period by setting $q_{r^*it} = q_{r^*i,t-1}$. Alternatively, 2. if the minimum block length has been met (line 5), a new block with a different level of resource allocation can begin. In this situation, the following three cases are considered:

(i) (lines 6, 7) If there is sufficient resource availability, then the current resource block is extended by setting $q_{r^*it} = q_{r^*i,t-1}$.

(ii) (lines 8-16) If enough resource is available to satisfy the minimum resource allocation to activity i for at least one minimum block length, then a new block can be started. The maximum principle resource allocation to this block is limited by the following considerations: 1. a new block must satisfy the minimum block length, and hence principle resource allocation cannot exceed $\xi_i/l^{\min}$; 2. the limited resource availability during the time periods $t, \ldots, t + l^{\min} - 1$ cannot be exceeded;

3. the greediness parameter ρ_i limits principle resource allocation through the function $w_i/(\underline{d}_i + \rho_i)$. Additionally, if t is before the earliest finish time of i, the principle resource allocation is also limited by $\varphi_i/(EF_i - t)$ in order to reduce unnecessary over-allocation. If a new block of increased resource allocation can begin, an additional check is made in line 15 to ensure that the increased resource allocation does actually speed-up processing. If not, then the current resource block is continued.

(iii) (lines 17, 18) If there is not enough resource available to continue the current block, nor satisfy the minimum resource allocation for one minimum block length, then the activity can not be completed, and the activity is rescheduled to start at the next resource feasible time period.

Having determined the principal resource allocation to activity i at time t, if fewer than two minimum block lengths remain until activity i is completed under the current resource allocation, over-allocation of resource to activity i is limited by setting $q_{r^*it} = \max\left(\underline{q}_{r^*i}, \xi_i/\lceil\frac{\xi_i}{q_{r^*it}}\rceil\right)$ in line 20. Finally, in line 21, the dependent resource allocation is updated based on the principal resource allocation.

The FSGS ensures that each activity is scheduled feasibly with respect to the project resource constraints. This however may lead to violations of particular maximal time-lags. In this case, an unscheduling step can be performed in an attempt to restore feasibility. The unscheduling step shown in Algorithm 3.2 is invoked if the latest start time of activity i is missed in line 1 of Algorithm . The unscheduling step can easily be adapted for the alternative situation where it is the latest finish of activity i that is missed.

In Algorithm 3.2, t denotes the time period in which the unscheduling step was

Algorithm 3.1: Steps of FSGS for scheduling activity $i \in \boldsymbol{\lambda}$.

```
t = (earliest time and resource feasible starting period for activity $i$) + $\sigma_i$
**while** $l_{it} < l^{\min}$ *or* $\xi_i > 0$ **do**
    **if** $l_{it} < l^{\min}$ **then**
        $q_{r^*it} = q_{r^*i,t-1}$
    **else**
        **(i) if** $\varphi_{rt} \geq q_{ri,t-1}$ *for all* $r \in R$ **then**
            $q_{r^*it} = q_{r^*i,t-1}$
        **(ii) else if** $\varphi_{r\tau} \geq \underline{q}_{ir}$ *for all* $r \in R$, $\tau \in \{t, \dots, t + l^{\min} - 1\}$ **then**
            **if** $t < EF_i$ **then**
                max allocation =
                  $\min\left\{\frac{\xi_i}{l^{\min}}, \min_{r \in R,\, \tau \in \{t,\dots,t+l^{\min}-1\}} \left(\frac{\varphi_{r\tau}-\beta_{ri}}{\alpha_{ri}}\right), \frac{w_i}{\underline{d}_i+\rho_i}, \frac{\xi_i}{EF_i - t}\right\}$
            **else**
                max allocation =
                  $\min\left\{\frac{\xi_i}{l^{\min}}, \min_{r \in R,\, \tau \in \{t,\dots,t+l^{\min}-1\}} \left(\frac{\varphi_{r\tau}-\beta_{ri}}{\alpha_{ri}}\right), \frac{w_i}{\underline{d}_i+\rho_i}\right\}$
            $q_{r^*it} = \max\left(\underline{q}_{r^*i}, q_{r^*it}, \text{max allocation}\right)$
            **if** $\varphi_{r\tau} > q_{ri,t-1}$ *for all* $r \in R$, $\tau \in \{t, \dots, t + l^{\min} - 1\}$ **then**
                **if** $\lceil \frac{\xi_i}{q_{r^*it}} \rceil < l^{\min}$ *or* $\lceil \frac{\xi_i}{q_{r^*it}} \rceil > \lceil \frac{\xi_i}{q_{r^*i,t-1}} \rceil$ **then**
                    $q_{r^*it} = q_{r^*i,t-1}$
        **(iii) else**
            restart $i$ at next resource feasible time period
        **if** $\lceil \frac{\xi_i}{q_{r^*it}} \rceil < 2 \cdot l^{\min}$ **then**
            $q_{r^*it} = \max\left(\underline{q}_{r^*i}, \xi_i / \lceil \frac{\xi_i}{q_{r^*it}} \rceil\right)$
    $q_{rit} = \alpha_{ri} q_{r^*it} + \beta_{ri}$ for all $r \in R^{\text{dep}}$
```

invoked and S denotes the set of scheduled activities. The unscheduling step begins by computing the sets of activities which have starts and finishes that determine the latest start time of activity i in lines 1 and 2. These sets are denoted by U_i^s and U_i^f respectively. The set of activities that must be unscheduled, denoted by U_i, is computed in line 3 as the activities in U_i^s and U_i^f, plus any activities $j \in S$ with a start time later than the earliest start time of any activity in U_i^s or U_i^f. Following the unscheduling of the activities in U_i, the earliest start times of the activities in U_i^s and the earliest finish times of the activities in U_i^f, are increased by the amount by which they were missed, and the activity time-windows are updated based on the activities that remain in S. The unscheduled activities are then rescheduled using Algorithm 3.1.

Algorithm 3.2: Unscheduling step for when LS_i is missed at time t.

```
1  $U_i^s = \{j \in S,\ (i,j,b_1) \in E_{SS} : t > s_j - b_1\}$;
2  $U_i^f = \{j \in S,\ (i,j,b_2) \in E_{SF} : t > f_j - b_2\}$;
3  $U_i = \{j \in S : s_j \geq \min_{k \in U_i^s \cup U_i^f} s_k\}$;
4  Unschedule activities $j \in U_i$ and update $S$;
5  for $j \in U_i^s$ do
6      $ES_j = s_j + (t - LS_i)$;
7      Update $EF_j$, $LS_j$, $LF_j$ based on $S$;
8  for $j \in U_i^f$ do
9      $EF_j = f_j + (t - LS_i)$;
10     Update $ES_j$, $LS_j$, $LF_j$ based on $S$;
11 for $j \in N \setminus (S \cup U_i^s \cup U_i^f)$ do
12     Update $ES_j$, $EF_j$, $LS_j$, $LF_j$ based on $S$;
13 for $j \in U_i \cup \{i\}$ do
14     Schedule $i$ using Algorithm 3.1;
```

We refer to the FSGS which makes use of the unscheduling step as the FSGS-U. FSGS-U is a recursive and computationally expensive procedure and therefore a maximum number of unscheduling attempts is specified upon implementation. If this maximum is reached, then the schedule is completed without the use of the unscheduling step. Hence, although the FSGS and FSGS-U always produce resource feasible schedules, neither heuristic can guarantee that these schedules are also time-feasible.

Example

To illustrate the FSGS and unscheduling step outlined above, we provide an example of their application to the GFRCPSP instance shown in Figure 4.1. Figure 4.3 shows the schedule outputted by Algorithm 3.1 using the parameters $\boldsymbol{\lambda} = (2, 1, 3, 5, 4)$, $\boldsymbol{\rho} = (0, 0, 0, 0, 1)$, and $\boldsymbol{\sigma} = (0, 0, 0, 0, 1)$ after performing the following steps:

- Activity 2 ($\sigma_2 = 0$, $\rho_2 = 0$) is started at time 0 and allocated its maximum resource limit of 4 units. It finishes at time 3.

- Activity 1 ($\sigma_1 = 0$, $\rho_1 = 0$) is started without delay at time 0 and allocated the remaining 2 available units of resource. At time 3, its resource allocation can be increased. The 9 units of resource it still requires are spread over the 2 periods to its earliest finish time of 5, resulting in an allocation of $9/2$ units per period.

- Activity 3 ($\sigma_3 = 0$, $\sigma_3 = 0$) is started at its earliest feasible start time of 5 and allocated its maximum of 4 units of resource. It finishes at time 8.

- Activity 5 ($\sigma_5 = 1$, $\rho_5 = 1$) is scheduled with one period of enforced delay at time 6 and the 2 remaining resource units at time 6 and 7 are allocated to it. Its resource allocation can increase once activity 3 ends. Since activity 5's earliest finish time is 11, 11/3 units of resource are allocated over the remaining 3 time periods.

- Activity 4 ($\sigma_4 = 0$, $\sigma_4 = 0$) is attempted to be scheduled at its earliest start time of 8. However, this violates its latest start time is 6, caused by the maximum time-lag of length 3 from the finish of activity 2.

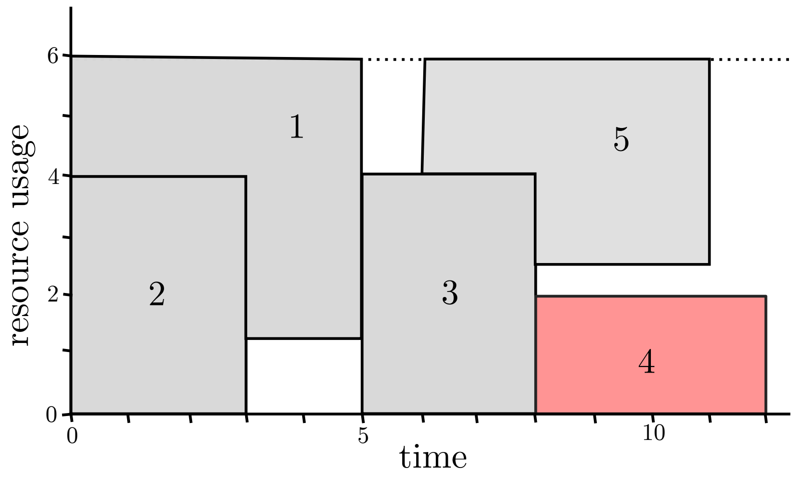


Figure 4.3: Schedule to the GFRCPSP instance shown in Figure 4.1 generated by Algorithm 3.1 with the input parameters $\boldsymbol{\lambda} = (2, 1, 3, 5, 4)$, $\boldsymbol{\rho} = (0, 0, 0, 0, 1)$, and $\boldsymbol{\sigma} = (0, 0, 0, 0, 1)$. This schedule violates a maximum time-lag between the finish of activity 2 and the start of activity 4.

Since a maximum time-lag has been violated at time $t = 8$, Algorithm 3.1 invokes the unscheduling step shown in Algorithm 3.2. Activity 2 is the only activity with a precedence relationship that affects the latest start of activity 4 and so $U_4^s = \emptyset$ and $U_4^f = \{2\}$. All activities have start times greater than or equal to the start time of 2 and so $U_4 = \{1, 2, 3, 4, 5\}$. These are unscheduled, and the earliest start time of activity 2 is updated to $ES_2 = 0 + (8 - 6) = 2$. The earliest and latest start and finish time of the other activities are reset to their original values. Algorithm 3.1 is then used to reschedule all of these activities.

As before, activity 2 is scheduled first, although this time its earliest feasible start time is $ES_2 = 2$ and it finishes at time 5. Activity 1 can be started at time 0 with its maximum resource limit of 5. Its resource allocation is decreased to its minimum required allocation of 2 units at time 2, and it finishes at time 5. Activity 3 is started immediately after the finish of activity 2, at time 5 and it finishes at time 8. Considering its enforced delay, activity 5 starts at time 6 and uses the 2 available units of resource. At time 8, the resource allocation to activity 5 is increased to 11/3 and it finishes at time 11. Finally, activity 4 is scheduled to start at time 8, and since activity 2 finishes at time 5 its maximum time-lag is now satisfied. Activity 4 finishes at time 12. This schedule is now feasible, and is shown in Figure 4.4.

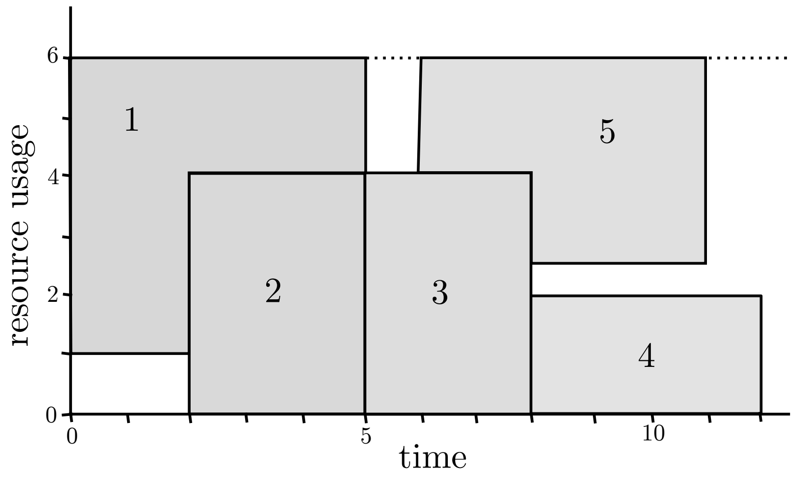


Figure 4.4: Schedule produced by applying the unscheduling step in Algorithm 3.2 to the schedule shown in Figure 4.3.

4.4.2 Genetic algorithm

We propose the following genetic algorithm (GA) to search over the space of input parameters to the FSGS described in the previous section. As explained above, an individual solution consists of an activity list representation $\boldsymbol{\lambda}$, a list of greediness parameters $\boldsymbol{\rho}$, and a list of delay parameters $\boldsymbol{\sigma}$.

An initial population of solutions is constructed by generating activity list representations $\boldsymbol{\lambda}$ randomly, subject to the topological ordering that results from the project precedence constraints. The greediness and delay parameters for each solution in the initial population are all set to zero. Hence, the FSGS will behave greedily in the initial population.

The 'fitness' of a feasible solution is measured by its makespan, whilst the fitness of an infeasible solution is measured as the total number of time periods by which precedence relationships are missed in that schedule, i.e. $\sum_{i \in N} \max(0, s_i - LS_i) + \max(0, f_i - LF_i)$, plus some fixed infeasibility penalty.

New individuals are obtained using the adapted two-point crossover of Franck et al. (2001), which takes two parent solutions as input, i.e. a 'father' and 'mother', and outputs two offspring solutions, i.e. a 'son' and a 'daughter'. Considering the son's activity list, activities before the first randomly selected crossover point are inserted in the order that they appear in the father's activity list. Activities between the first and second crossover points are inserted in the order that they appear in the mother's activity list, and the remaining activities after the second crossover are taken again from the father's activity list. If however, one of the crossover points fall in the middle of a cycle structure, the remaining activities of the cycle structure are inserted immediately into the son's activity list in the order that they appear in the relevant parent's activity list. This crossover is designed to keep activities that are related by a maximal time-lag close together in the resulting activity list, thus increasing the likelihood of generating feasible offspring. On the greediness and delay parameter lists, this crossover simply behaves like the standard two-point crossover of Hartmann (1998). The daughter solution is produced by reapplying the crossover with the roles of father and mother reversed.

The mutation operator of Hartmann (1998) is applied to the activity of each offspring solution. If feasible with respect to the project precedence relationships, this operator swaps each activity with the activity that follows it with probability p_λ. For each $i \in N$, ρ_i is mutated with probability p_ρ, and σ_i is mutated with probability p_σ. When ρ_i is mutated, half of the time it is either increased or decreased by one (with equal probability), and the other half of the time, a new value for ρ_i is chosen randomly from within its bounds. When σ_i is mutated, it is replaced by a random integer from within its bounds.

Having doubled the original population size, the top n_{elite} solutions are carried into the next generation. Following this, n_{rand} randomly generated new solutions are carried forward. The remaining spaces in the next generation are then chosen using 3-tournament selection from the current generation. The next generation has the same size as the initial population.

4.5 Computational study

This section compares the performance of the algorithms for solving the GFRCPSP we have presented in this chapter. We begin by briefly summarising each of the algorithms that have been included in this computational study:

- **MIP**: The MIP presented in Section 4.3 and based on the FP-DT3 model from Naber and Kolisch (2014).
- **GGA**: A greedy FSGS without the unscheduling step, embedded into the GA from Section 4.4.2 which searches only over activity list representations.
- **FGA**: The FSGS from Section 4.4.1 without the unscheduling step, embedded into the GA from Section 4.4.2.
- **FURS**: The full FSGS-U from Section 4.4.1, applied to randomly sampled solutions.
- **FUGA**: The full FSGS-U as described in Section 4.4.1, embedded into the GA from Section 4.4.2.
- **FGAU**: The same as FGA, however at the end of each generation, $n_{\text{reschedule}}$ of the least infeasible solutions are rescheduled using the full FSGS-U. These solutions are recorded but not added to the population.

By comparing the results of GGA, FGA and FUGA, the benefit of the FSGS and the unscheduling step can be evaluated. Similarly, by comparing the results of FURS and FUGA, the impact of the genetic algorithm can be assessed.

The population size for each metaheuristic algorithm was set to 500. The selection parameters used by the GA were $n_{elite} = 10$, and $n_{rand} = 30$, leaving $500 - n_{elite} - n_{rand} = 460$ individuals to be chosen from the previous generation through 3-tournament selection. The mutation rates used by the GA were $p_\lambda = 5\%$, $p_\rho = 5\%$, and $p_\sigma = 0.5\%$. Whenever the FSGS-U was used, the unscheduling step was invoked up to a limit of 20 times, after which the remaining activities were scheduled without the use of the unscheduling step. The metaheuristic algorithms were each given a limit of 50,000 schedules before their best found solutions were reported, whilst the MIP was solved using a time limit of 2 hours. All of the solution methods were written in Python 3.6.8 and run on a single thread of a 2.3GHz Intel Xeon E5-2699 v3 processor. The MIP was solved using Gurobi 8.1.0.

Before being solved by any of the methods, a basic pre-processing procedure was applied to each instance to compute the earliest and latest start and finish times of each activity. This procedure is outlined in detail in Appendix A.

4.5.1 Test instances

The six algorithms described in the previous section have been applied to five sets of GFRCPSP test instances of differing size and difficulty. The instances in these test sets have been created by extending the ProGen/max project generator of Schwindt (1996) to generate instances that allow for flexible resource allocation to activities.

The five developed test sets each contain 90 test instances, resulting in a total of 450 instances. The instances in these five test sets contain 10, 20, 30, 50 and 100 activities respectively, and are correspondingly referred to as P10, P20, P30, P50 and P100.

The order strength (OS), resource strength (RS) and resource factor (RF) parameters (Kolisch and Sprecher, 1997) have been controlled when generating these GFRCPSP instances. OS controls the number of precedence relationships in the project network, RS measures the restrictiveness of the resource availability, and RF indicates the average number of resources required by each activity. Since RS has the largest effect on the difficulty of solving an instance, only this parameter has been varied across the instances generated in each test set, with 30 instances being generated for each of the RS values 0.05, 0.15, 0.25. For every instance, OS is set to 0.4, and RF is set to 0.75. Each instance involves five resources, and the minimum block length is chosen randomly to be either 2, 3 or 4.

The five sets of test instances used in this computational study and the code used to generate them can be found at `https://github.com/boldm1/GFRCPSP_instgen`.

4.5.2 Computational results

This section reports and analyses the results from applying the six algorithms listed above to the P10, P20, P30, P50, P100 test sets.

We first report the number of instances for which a feasible solution was found by each algorithm. This is shown in Table 4.3 for each test set. Observe that FUGA is the only algorithm to find a feasible solution to all 450 test instances.

Test set	MIP	GGA	FSGA	FURS	FUGA	FGAU
P10	90	90	90	90	90	90
P20	90	89	90	90	90	90
P30	87	86	90	90	90	90
P50	35	86	90	90	90	90
P100	32	60	75	73	90	89

Table 4.3: Number of instances for which a feasible solution was found.

We now examine the results shown in Figure 4.5. Each row of plots corresponds to a different set of test instances, as labelled down the left-hand-side. The left-most plots shows the average percentage gap to the best known solution for each heuristic algorithm, as a function of the number of schedules that have been searched. When an algorithm failed to find a feasible solution to a given instance, a 'penalty gap' of 50% was applied. If an algorithm fails to find a feasible solution to at least one instance in a test set, its corresponding line is dashed to emphasise that the plotted value is an estimate that is dependent on this choice of penalty gap. The middle plots show the number of instances for which each heuristic algorithm finds the best known solution, as a function of the number of schedules that have been searched. Finally, the right-most plot of each row shows the performance profile of each solution method; that is, a function showing the proportion of instances solved to within τ of the best known solution.

Looking at the first two plots of each row, it is clear that FUGA is the strongest performing heuristic algorithm across all test sets, with its superiority becoming more pronounced as the instances become larger. Across all the test sets, the algorithms that make use of the unscheduling step (FURS, FUGA and FGAU) find the better solutions in the early generations than the algorithms that do not (GGA and FGA). However, only the algorithms that also use the GA (FUGA and FGAU) maintain this advantage over the 50,000 schedules.

Looking at the performance profiles, we can directly compare the MIP with the heuristic algorithms. The MIP outperforms FUGA and the other heuristic algorithms for the P10 test set, where it finds a provably optimal solution for all 90 instances. The MIP remains the best performing algorithm for P20, where it solves 77/90 instances to optimality. Despite also finding an optimal solution to 77/90 instances for P30, the MIP fails to find a solution within 50% of the best known solution for 9/90 instances, whereas FUGA finds a solution within 5% of the best known solution for 86/90 instances. For P50 and P100, the performance

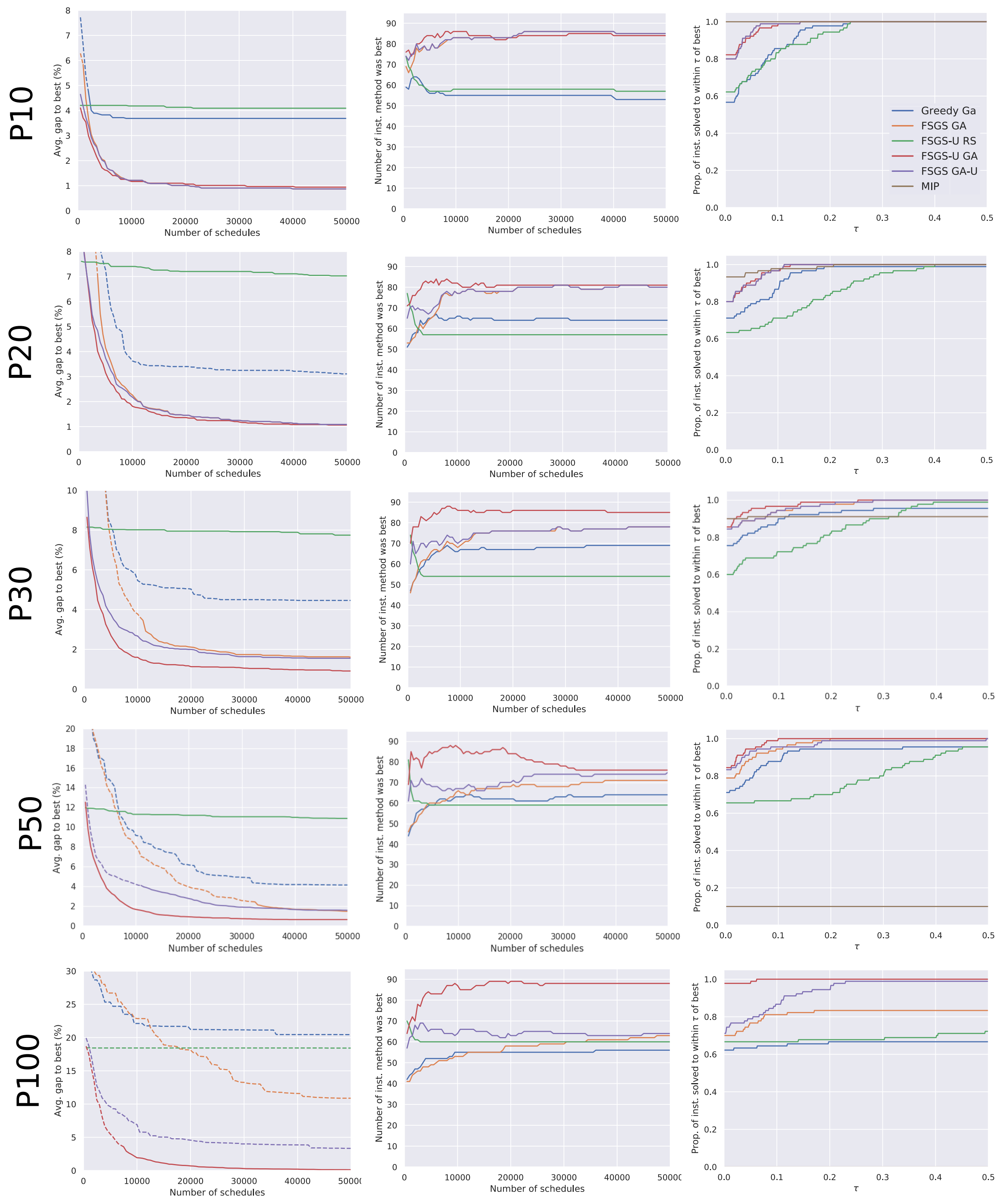


Figure 4.5: Left-most plots show average percentage gap to best known solution for each heuristic algorithm, as a function of the number of schedules searched. Middle plots show the number of instances for which each heuristic algorithm finds the best solution, as a function of the number of schedules searched. Right-most plots show performance profile of each algorithm.

of MIP dramatically worsens when compared with the heuristic algorithms.

Table 4.4 compares the average percentage gap to the critical-path-based lower bound of solutions found by the MIP and the strongest performing heuristic algorithm, FUGA, across the five test sets. The instances in each set have been separated by their resource strength (RS). The instances with smaller RS values have more restrictive resource availabilities and are therefore more challenging. To enable a fair comparison of the solution methods, the instances for which both algorithms found a feasible solution have been presented separately from the instances for which only FUGA finds a feasible solution. There were no instances for which only the MIP found a feasible solution. Looking at these results confirms that the MIP found the better solutions over P10 and P20, but that its perfor-

		MIP & FUGA			Only FUGA	
Test set	RS	#	Δ_{lb}^{MIP}	Δ_{lb}^{FUGA}	#	Δ_{lb}^{FUGA}
	0.05	30	10.90	13.14	0	-
P10	0.15	30	2.17	3.24	0	-
	0.25	30	0.49	0.49	0	-
	0.05	30	19.37	19.65	0	-
P20	0.15	30	0.25	0.49	0	-
	0.25	30	0.00	0.00	0	-
	0.05	28	150.81	13.43	2	10.71
P30	0.15	29	0.00	0.00	1	0
	0.25	30	14.56	0.00	0	-
	0.05	0	-	-	30	25.24
P50	0.15	6	617.23	0.00	24	0.00
	0.25	29	769.84	0.00	1	0.00
	0.05	0	-	-	30	18.81
P100	0.15	3	1029.66	0.00	27	0.00
	0.25	29	1442.55	0.00	1	0.00

Table 4.4: Average percentage gap to the critical path-based lower bound of solutions found by the MIP and FUGA. These values are denoted by Δ_{lb}^{MIP} and Δ_{lb}^{FUGA}, respectively.

mance dramatically worsens over the larger test sets. In contrast, the quality of the solutions found by FUGA remained roughly constant across all five sets.

We now consider the computational speed of the heuristic algorithms. Table 4.5 shows the average time required by each heuristic algorithm to generate 1000 schedules for each type of instance across the test sets. The average times over all the instances in each test set are shown in bold. The two algorithms that do not use the unscheduling step have computation times that scale at a constant factor of roughly 2.5 as the instance size is doubled. The algorithms that do make use of the unscheduling step scale less well. When RS = 0.05, the unscheduling step is required far more frequently, and as a result the algorithms that use the unschedul-

Test set	RS	GGA	FGA	FURS	FUGA	FGAU
P10	0.05	4.8	2.9	11.4	5.0	2.9
	0.15	4.9	2.7	7.9	5.4	2.8
	0.25	5.9	2.8	5.7	4.0	2.8
		5.2	**2.8**	**8.3**	**4.8**	**2.8**
P20	0.05	11.0	8.0	48.6	21.6	8.2
	0.15	14.1	6.8	22.3	18.2	6.9
	0.25	10.0	6.5	16.4	14.1	6.6
		11.7	**7.1**	**29.1**	**17.9**	**7.2**
P30	0.05	16.9	16.9	69.8	24.1	15.1
	0.15	16.5	18.2	41.8	22.5	16.4
	0.25	16.5	18.6	29.3	15.3	14.2
		16.6	**17.9**	**47.0**	**20.7**	**15.2**
P50	0.05	45.3	37.4	185.0	98.0	41.5
	0.15	37.8	35.3	125.4	55.7	38.3
	0.25	27.9	29.3	120.0	42.1	27.5
		37.0	**34.0**	**143.5**	**65.3**	**35.7**
P100	0.05	96.3	87.4	567.9	469.3	123.2
	0.15	92.5	88.4	527.0	331.1	102.4
	0.25	101.3	95.9	503.3	157.6	100.2
		96.7	**90.6**	**532.8**	**319.3**	**108.6**

Table 4.5: Average time to generate 1000 schedules in seconds.

ing step run particularly slowly across these instances. The FGAU algorithm has been developed in an attempt to make use of the benefits of the unscheduling step whilst keeping computational times low. FGAU scales only slightly less well than the algorithms without the unscheduling step and maintains solution quality to a good extent, finding a feasible solution to all but one of the 450 instances.

4.6 Case Study: The Sellafield nuclear decommissioning project

In this section we outline the application of the model and solution methods presented in this chapter to the Sellafield decommissioning project. Located on the Cumbrian coastline in North-west England, the Sellafield nuclear site covers 6 square kilometers, contains more than 200 nuclear facilities, and is the location of the world's the largest inventory of untreated nuclear waste. Since the second world war, the site has been a centre of the nuclear industry in the UK, being home to the UK's original nuclear weapons program, the world's first commercial nuclear power station, and the UK's nuclear reprocessing operations. With Sellafield coming to the end of its useful lifespan, focus is now turning to its decommissioning and the safe clean-up of legacy nuclear waste. This is a project that is expected to take in excess of 100 years to complete and cost over £90 billion (NDA, 2019).

Given the scale and complexity of the Sellafield decommissioning project, it is essential that it is scheduled carefully and systematically. It is a requirement that any decommissioning schedule is feasible with respect to the precedence relationships that exist between the site facilities, as well as with respect to limits on yearly budget, human resource, and the amount of waste material that can safely be processed each year. In the following section, we present the decommissioning problem in more detail, and explain how it can be modelled as a GFRCPSP.

4.6.1 Modelling the decommissioning problem

Having completed its operations, each nuclear facility on the Sellafield site must go through a programme of decommissioning. In particular, there are three decommissioning stages that a building might have to complete: Post-Operation Clean-Out (POCO), Interim Decommissioning (ID), and Final Decommissioning (FD). Each building must complete some or all of these stages in order to be fully decommissioned. Importantly, a number of new buildings need to be constructed in order to support the decommissioning of certain existing facilities. For example, over-structures must be built to contain the leakage of nuclear material from the demolition of certain nuclear facilities. These supporting buildings must complete an additional *construction* (Cons) phase before they become operational.

All activities require a total amount of money and man-hours to be completed. In addition to this, the ID and FD activities also produce a total amount of waste that must be processed. This waste is modelled as a resource that must be allocated to these activities. Two types of waste are considered: Intermediate-Level Waste (ILW) and Low-Level Waste (LLW). For each activity, man-hours is the principal resource which determines the allocation of the other three resources.

The overall total cost, total number of man-hours required and total waste produced by each activity are known. In addition to these overall resource requirements, upper and lower bounds on the yearly allocation of each of these resources

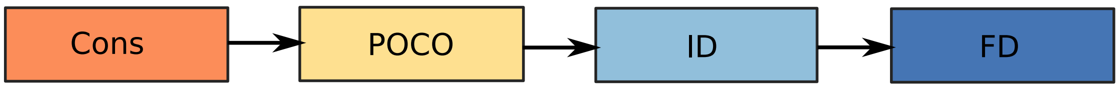


Figure 4.6: The construction and decommissioning stages of a facility on the Sellafield site. Note that, whilst not every facility must complete all of these stages, the stages that a facility does need to complete must be executed in the order dictated by this figure.

to each activity are specified. These upper and lower bounds usually arise due to the physical constraints of the site which affect, for example, the maximum number of workers that can safely fit into one facility, or the maximum amount of waste that a single worker can remove from a given facility.

With some individual decommissioning activities expected to last upwards of 50 years, it is unrealistic to assume a fixed and unchanging resource allocation for the entire duration of every activity. Instead, by modelling the decommissioning project as a GFRCPSP we allow resources to be flexibly allocated to each activity, subject to the upper and lower bounds on the annual resource allocation. Additionally, a minimum block length of 2 years is also enforced. In the absence of data on resource availability, the availability of the principal 'man-hours' resource was assumed to be equal to the maximum upper bound on its per period allocation over all activities. The resource availability of each of the non-principal resources was chosen to be non-limiting.

The activities associated with each building must be completed in the order dictated by the decommissioning precedence relationships shown in Figure 4.6, as well as a set of strict precedence relationships between facilities that result from the physical layout of the site. Some examples of the types of relationship that

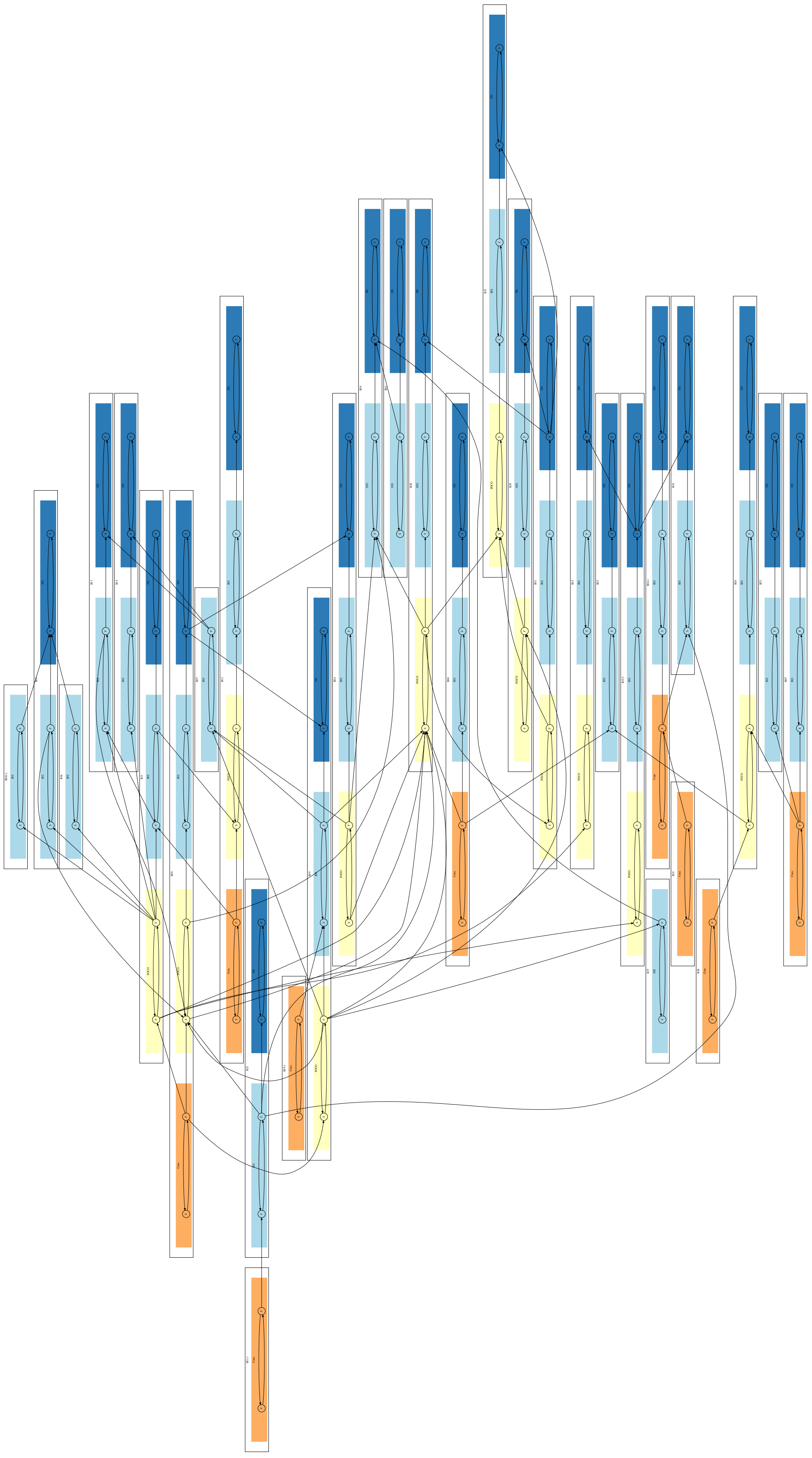


Figure 4.7: The largest connected component of the precedence network of the Sellafield decommissioning project. Each separate decommissioning activity has a start and finish node contained in a box coloured according to the colours shown in Figure 4.6. The activities associated with a single building follow the decommissioning order shown in Figure 4.6, and are shown on the same rank within a larger box. Note that time-lags are not shown on this network diagram.

exist between the different facilities on the site are as follows:

- Buildings A and B share a structural wall and therefore must start ID at the same time. This is modelled as two zero time-lag start-to-start precedence relationships going in opposite directions;
- Building C must have been operational for at least a year before it can begin to process the waste produced by the ID phase of Building D. This is modelled as a 1-year minimal time-lag relationship from the finish of Building C Cons to the start of the Building D ID;
- The operation of Building E relies on the operation of Building F, and hence Building F must have finished Cons before Building E finishes Cons. This is modelled as a zero time-lag finish-to-finish relationship;
- Building F is structurally sound until 2050 and therefore must have completed ID by this date. This deadline is modelled as a maximum time-lag between the dummy project-start activity, and the finish of Building F ID;

Note how these relationships require the use of generalised precedence relationships in order to be accurately captured by the model.

The specific problem instance solved in this case study corresponds to a subset of the Sellafield site involving 43 buildings and a total of 95 separate decommissioning activities. The largest connected component of the resulting project precedence network is shown in Figure 4.7. The following section describes the results of solving this problem using each of the solution approaches proposed in this chapter.

4.6.2 Results

Firstly, the instance was formulated using the MIP in Section 4.3 and passed to the Gurobi 8.1.0 solver. As was the case for the harder instances from the P100 test set, this approach failed to find a feasible solution to the decommissioning scheduling problem within the time limit of two hours.

In contrast, all five heuristic algorithms found a feasible decommissioning schedule. Figure 4.8 shows the best solution found by each of the algorithms as a function of time (in seconds). Each metaheuristic was given a search limit of 50,000 schedules and the point at which each line stops in Figure 4.8 gives the time required by the corresponding algorithm to reach this limit.

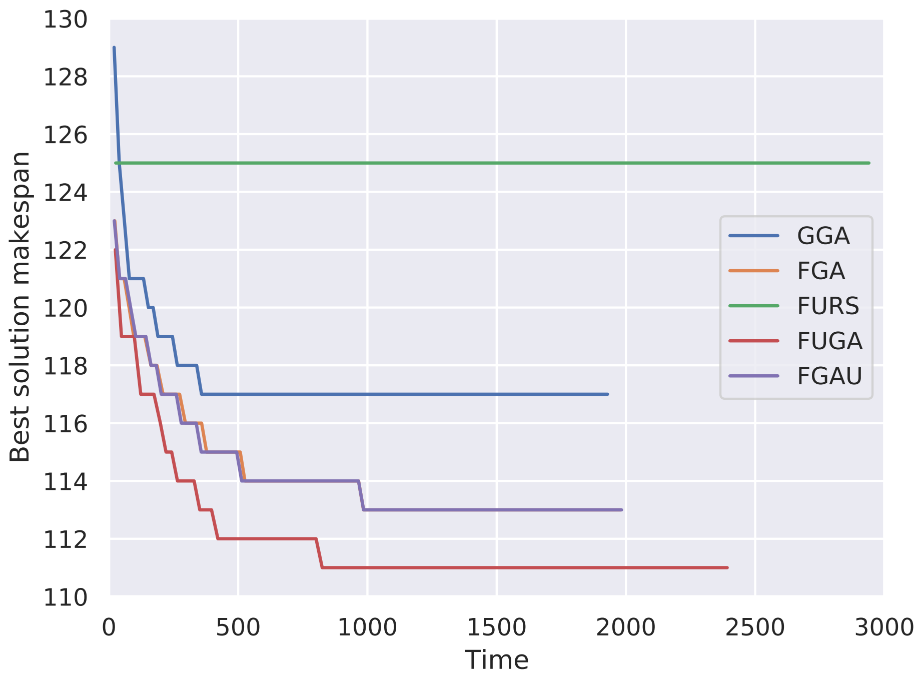


Figure 4.8: Best found makespan for each metaheuristic algorithm, as a function of time (in seconds). Each algorithm was used to generate 50,000 schedules.

The quality of the solutions found by the different solution methods follows the same order as the results presented in Section 4.5.2. FURS is the weakest algorithm, finding a feasible schedule with makespan of 125 years, followed by GGA, which finds a schedule with makespan of 117 years. The FGA and FGAU algorithms both find schedules that complete in 113 years. As expected, FUGA is the strongest performing algorithm, finding a schedule with a makespan of 111 years. It is also worth noting that, in this instance, the use of the unscheduling step in FUGA does not dramatically affect its running time, with FUGA requiring only 20.8% longer than FGA to generate 50,000 schedules. The reason for this is that after the use of the unscheduling step in the early generations, the solution population quickly fills with feasible solutions resulting in it being required much less frequently throughout later generations.

This case study provides a practical motivation for the use of the GFRCPSP, and further demonstrates the effectiveness of the proposed FUGA algorithm for solving large-scale GFRCPSP instances.

4.7 Conclusion

Motivated by the need to model a real-world large-scale nuclear decommissioning project, this chapter has introduced the GFRCPSP. This new project scheduling model serves as an extension to the RCPSP that combines the inclusion of both generalised precedence constraints and flexible resource allocation. Firstly, an MIP formulation has been proposed to solve this problem. Following this, a metaheuristic solution approach has been developed for solving larger instances for which the MIP fails to find a solution. The proposed FUGA algorithm is based on a non-greedy flexible serial schedule generation scheme with an unscheduling step, embedded in a genetic algorithm that searches over the space of inputs to the scheduling heuristic. This heuristic approach produces the strong results when compared with the MIP and four other benchmarking heuristics in a computational study. Given the relatively slow computation time of proposed metaheuristic algorithm, the FGAU variant has been suggested in an attempt to use the computationally expensive unscheduling step more sparingly. FGAU retains a good solution quality in most instances and significantly reduces computation times as desired. Finally, the application of the GFRCPSP to model the Sellafield nuclear decommissioning project highlights the relevance of this new model, and further demonstrates the effectiveness of the FUGA algorithm.

In terms of further research directions, it seems worthwhile to investigate further improvements to the metaheuristic algorithms proposed here. An improved algorithm would likely employ a hybrid metaheuristic strategy, extending the proposed genetic algorithm by including an additional local search heuristic.

Chapter 5

A Compact Reformulation of the Two-Stage Robust Resource-Constrained Project Scheduling Problem

5.1 Introduction

The resource-constrained project scheduling problem (RCPSP) consists of scheduling a set of activities, subject to precedence constraints and limited resource availability, with the objective of minimising the overall project duration, known as the makespan. Given its practical relevance to a number of industries, including construction (Kim, 2013), manufacturing (Gourgand et al., 2008), R&D (Vanhoucke, 2006), and personnel scheduling (Drezet and Billaut, 2008), the RCPSP and many of its variants have been widely studied since a first model was introduced by Pritsker et al. (1969). The vast majority of this research, however, has examined the RCPSP under the assumption that the model parameters are known deterministically (for a survey of the deterministic RCPSP, see Artigues et al. (2008)), but clearly, in practice, large projects are subject to non-trivial uncertainties. For instance, poor weather might delay construction times, uncertain delivery times of parts may delay manufacturing activities, and the durations of research activities are inherently uncertain. As a result, in recent years, increasing attention has been given to the uncertain RCPSP, where scheduling decisions must be made whilst activity durations are unknown.

There exist two main approaches for solving the uncertain RCPSP. The first is to view the problem as a dynamic optimisation problem where scheduling decisions are made each time new information becomes available according to a scheduling policy (Igelmund and Radermacher, 1983a,b; Möhring and Stork, 2000). Most recently, Li and Womer (2015) use approximate dynamic programming to find an adaptive closed-loop scheduling policy for the uncertain RCPSP.

The second approach aims to proactively develop a robust baseline schedule that protects against delays in the activity durations. Zhu et al. (2007) present a two-stage stochastic programming formulation for building baseline schedules for projects with a single resource. Bruni et al. (2015) present a chance-constraint-based heuristic for constructing robust baseline schedules and Lamas and Demeulemeester (2016) introduce a procedure for generating robust baseline schedules that is independent of later reactive scheduling procedures. For a review of both dynamic and proactive project scheduling, see Herroelen and Leus (2005).

Although frequently referred to as robust, none of the scheduling methods described above make use of robust optimisation in the sense of Ben-Tal and Nemirovski (1998, 1999, 2000). Over the last 20 years, robust optimisation has emerged as an effective framework for modelling uncertain optimisation problems.

Unlike stochastic programming, robust optimisation does not require probabilistic knowledge of the uncertain data. Instead, the robust optimisation approach only assumes that the uncertain data lie somewhere in a given uncertainty set, and then aims to find solutions that are robust for all scenarios that can arise from that uncertainty set.

The applicability of robust optimisation as a method for solving uncertain optimisation problems has increased following the introduction of adjustable robust optimisation (Ben-Tal et al., 2004; Yanıkoğlu et al., 2019). Adjustable robust optimisation extends static robust optimisation into a dynamic setting, where a subset of the decision variables must be determined under uncertainty, whilst other variables can be adjusted following observations of the uncertain data. As well as accurately modelling the decision process undertaken by many real-world decision-makers, adjustable robust optimisation overcomes the over-conservativeness that restricts the applicability of static robust optimisation models. For extensive surveys on robust optimisation, see Ben-Tal et al. (2009); Bertsimas et al. (2011); Gorissen et al. (2015); Goerigk and Schöbel (2016).

Despite the successful application of robust optimisation in many different fields (see Bertsimas et al. (2011)), few papers have directly applied robust optimisation in the construction of robust baseline project schedules. Balouka and Cohen (2021) consider the multi-mode RCPSP under the framework of robust optimisation, and present a solution approach based on Benders' decomposition (Benders, 1962). Artigues et al. (2013b) consider the RCPSP under uncertain activity durations and present an iterative scenario-relaxation algorithm, with the objective of minimising the worst-case absolute regret (Kouvelis and Yu, 1997).

Bruni et al. (2017) introduce the two-stage adjustable robust RCPSP that we consider in this chapter. For the case of budgeted uncertainty, they solve this problem using a Benders'-style decomposition approach. Bruni et al. (2018) extend this work and present a computational study of solution methods for solving the two-stage adjustable RCPSP. An additional Benders'-style algorithm is compared against a primal decomposition algorithm, as well as the algorithm presented in Bruni et al. (2017). The primal decomposition algorithm is shown to be the best performing algorithm for solving the two-stage adjustable RCPSP. At the same time, only 767 of 1440 instances could be solved to optimality, which means that a large gap of unsolved instances remains.

Simultaneously and independently of this work, Pass-Lanneau et al. (2020) have studied a related problem known as the anchor-robust RCPSP, where baseline schedules are developed such that the start times of the activities in a given subset are guaranteed to remain unchanged upon the realisation of the uncertain data. Examining their problem, they independently obtain a similar formulation to the one we present in Section 5.3.2.

The contributions of this chapter are as follows. We present a new compact reformulation of the two-stage adjustable robust RCPSP with budgeted uncertainty. This is the first compact formulation for this problem, allowing us to solve it directly using standard optimisation software. As a result, and as computational experiments confirm, our compact reformulation can be solved significantly faster, and for a much greater number of instances than the current best algorithm for solving this problem.

The remainder of this chapter is organised as follows: Section 5.2 introduces the two-stage adjustable robust RCPSP in detail, before Section 5.3 derives a compact reformulation of this problem and computational experiments are presented in Section 5.4. Concluding remarks are made in Section 5.5.

5.2 The two-stage robust RCPSP

A project consists of a set $V = \{0, 1, \ldots, n, n+1\}$ of non-preemptive activities, where 0 and $n+1$ are dummy source and sink activities with duration 0. Each activity $i \in V$ requires an amount $r_{ik} \geq 0$ of resource $k \in K$, where K is the set of project resource types. Each resource $k \in K$ has a finite availability R_k in each time period. Each activity $i \in V$ has a nominal duration given by $\bar{\theta}_i$, and a worst-case duration given by $\bar{\theta}_i + \hat{\theta}_i$, where $\hat{\theta}_i$ is its maximum deviation. In addition to resource constraints, the project activities must be scheduled in a manner that respects a set E of strict finish-to-start precedence constraints, where $(i, j) \in E$ enforces that activity i must have finished before activity j can begin. A project can be represented on a directed graph $G(V, E)$. An example project involving seven non-dummy activities and a single resource is shown in Figure 5.1.

We assume that the duration of each activity $i \in V$ lies somewhere between its nominal value $\bar{\theta}_i$ and its worst-case value $\bar{\theta}_i + \hat{\theta}_i$. Additionally, we follow Bertsimas and Sim (2004) and assume that only a subset of all activities can simultaneously attain their worst-case values. Hence, the set in which we assume durations can lie, known as the uncertainty set, is given by

$$\mathcal{U}(\Gamma) = \left\{ \theta \in \mathbb{R}_+^{|V|} \, : \, \theta_i = \bar{\theta}_i + \delta_i \hat{\theta}_i, \; 0 \leq \delta_i \leq 1 \; \forall i \in V, \; \sum_{i \in V} \delta_i \leq \Gamma \right\},$$

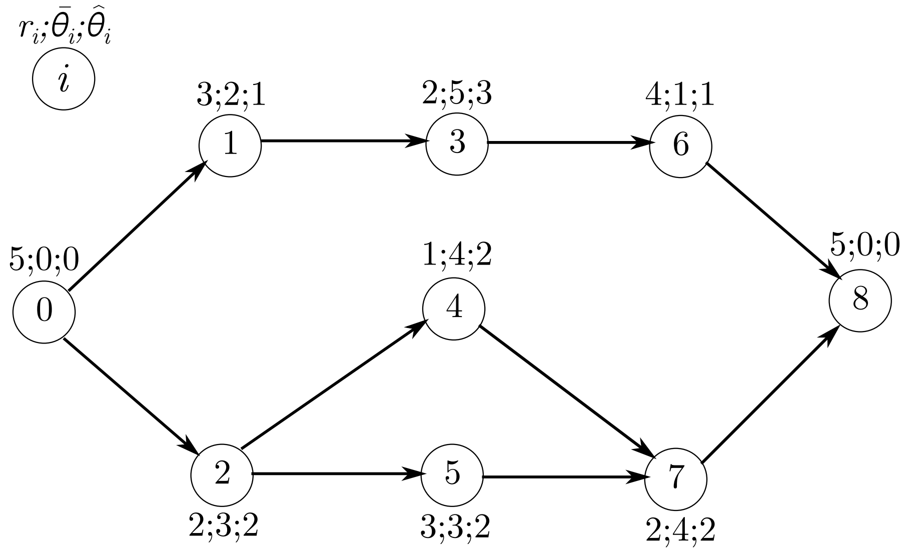


Figure 5.1: Example project involving seven non-dummy activities and a single resource with $R_1 = 5$.

where Γ determines the robustness of the solution by controlling the number of activities that are allowed to reach their worst-case duration simultaneously. For $\Gamma = 0$, each activity takes its nominal duration and the problem reduces to the deterministic RCPSP. At the other extreme, when $\Gamma = n$, every activity can take its worst-case duration, and this uncertainty set becomes equivalent to interval uncertainty.

The robust RCPSP lends itself naturally to a two-stage decision process, where resource allocation decisions need to be made at the start of the project before the uncertain activity durations become known, but the activity start times can be decided following the realisation of the activity durations. Hence, resource allocation decisions constitute the set of first-stage decisions, whilst the activity start times constitute the set of second-stage decisions.

More specifically, the first-stage resource allocation decisions consist of determining a feasible extension of the project precedence relationships E so that all resource conflicts are resolved. A forbidden set (Igelmund and Radermacher, 1983a) is any subset $F \subseteq V$ of non-precedence-related activities such that $\sum_{i \in F} r_{ik} > R_k$ for at least one $k \in K$, i.e. the activities of F cannot be executed simultaneously without violating a resource constraint. A minimal forbidden set is a forbidden set that does not contain any other forbidden set as a subset. We denote the set of minimal forbidden sets by $\mathcal{F}$. For the example project in Figure 5.1, $\mathcal{F} = \big\{\{1,5\},\{2,6\},\{5,6\},\{6,7\},\{3,4,5\}\big\}$. The resource conflict represented by each minimal forbidden set can be resolved by adding an additional precedence relationship to the project network. Bartusch et al. (1988) show that solving the RCPSP is equivalent to finding an optimal choice of additional precedence relationships $X \subseteq V^2 \setminus E$, such that the extended project network $G'(V, E \cup X)$ is acyclic and contains no forbidden sets. Such an extension X to the project precedence network is referred to as a sufficient selection. Hence, a solution to the first-stage problem corresponds to the choice of a sufficient selection X. Figure 5.2 shows the extended project network for a sufficient selection to the example project shown in Figure 5.1 (arcs in X are dashed).

Given the extended project network resulting from the choice of sufficient selection made in the first stage, the second stage problem consists of determining activity start times in order to minimise the worst-case makespan in this extended network. Since all resource conflicts have been resolved in the first-stage problem, the second stage problem contains no resource constraints.

Hence, the two-stage robust RCPSP under budgeted uncertainty is given by:

$$\min_{X \in \mathcal{X}} \max_{\theta \in \mathcal{U}(\Gamma)} \min_{S \in \mathcal{S}(X,\theta)} S_{n+1} \tag{5.1}$$

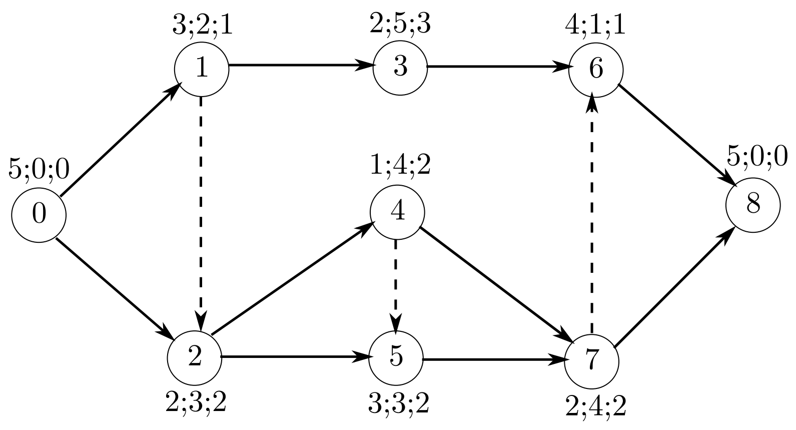


Figure 5.2: An extension of the example project shown in Figure 5.1, corresponding to the sufficient selection given by the dashed arcs.

where $\mathcal{X}$ is the set of sufficient selections, and $\mathcal{S}(X, \theta)$ is the set of feasible activity start times given the activity durations $\theta \in \mathcal{U}(\Gamma)$ and choice of sufficient selection X. That is,

$$\mathcal{S}(X, \theta) = \left\{ S \in \mathbb{R}_+^{|V|} \ : \ S_0 = 0, \ S_j - S_i \geq \theta_i \, \forall (i,j) \in E \cup X \right\}.$$

To solve this problem we propose a mixed-integer programming formulation, outlined in the following section.

5.3 A compact reformulation

In this section, we present a reformulation of the two-stage robust RCPSP. Unlike existing formulations for the two-stage adjustable RCPSP, the formulation we propose is *compact*, i.e. it contains polynomially many constraints and variables. We begin by first examining the adversarial sub-problem of maximising the worst-case makespan for a given sufficient selection.

5.3.1 The adversarial sub-problem

Suppose the solution to the first-stage problem provides a sufficient selection $X \in \mathcal{X}$, and is given by a vector $y \in \{0,1\}^{V \times V}$ where

$$y_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \cup X \\ 0 & \text{otherwise.} \end{cases}$$

The second-stage sub-problem that arises can be considered from the point of view of an adversary who wishes to choose the worst-case scenario of delays for the given first-stage solution. Following the adversary's choice of delays, we can determine the start time of each activity in order to minimise this worst-case makespan.

Let us assume a fixed scenario $\theta \in \mathcal{U}(\Gamma)$ given by the vector $\delta \in [0,1]^{|V|}$. In this case, the inner minimisation problem becomes

$$\min S_{n+1} \tag{5.2}$$

$$\text{s.t. } S_0 = 0 \tag{5.3}$$

$$S_j - S_i \geq \bar{\theta}_i + \delta_i \hat{\theta}_i - M(1 - y_{ij}) \qquad \forall (i,j) \in V^2 \tag{5.4}$$

$$S_i \geq 0 \qquad \forall i \in V, \tag{5.5}$$

where M is some number greater than or equal to the maximum possible minimum makespan. By taking the dual of (5.2)-(5.5), and then introducing the adversarial delay variables δ_i, $i \in V$, we obtain the following non-linear mixed-integer programming formulation for the adversarial sub-problem, first introduced in Bruni et al. (2017):

$$\max \sum_{(i,j)\in V^2} \left(\bar{\theta}_i + \delta_i\hat{\theta}_i - M(1-y_{ij})\right)\alpha_{ij} \tag{5.6}$$

$$\text{s.t.} \sum_{(i,j)\in V^2} \alpha_{ij} - \sum_{(j,i)\in V^2} \alpha_{ji} = 0 \qquad \forall j \in V \tag{5.7}$$

$$\sum_{(0,i)\in V^2} \alpha_{0i} = 1 \tag{5.8}$$

$$\sum_{(i,n+1)\in V^2} \alpha_{i,n+1} = 1 \tag{5.9}$$

$$\sum_{i\in V} \delta_i \leq \Gamma \tag{5.10}$$

$$0 \leq \delta_i \leq 1 \qquad \forall i \in V \tag{5.11}$$

$$\alpha_{ij} \in \{0,1\} \qquad \forall (i,j) \in V^2. \tag{5.12}$$

Observe that (5.6)-(5.9) correspond to the dual of (5.2)-(5.5), where the dual variables α_{ij} are continuous. Note also how the delay variables δ_i, $i \in V$, are related to the dual variables α_{ij} through the objective (5.6). The dual variables α_{ij} determine a longest path through the network defined by the first-stage variables y_{ij}. Hence, when $\alpha_{ij} = 1$, edge (i,j) is included in this longest path, and the duration of activity i, which determined by its delay variable δ_i, contributes to its length. Therefore, with the addition of the delay variables δ_i, $i \in V$, this adversarial sub-problem can be thought of as a non-linear longest-path problem through the network determined in the first-stage problem, where up to Γ units of delay can be distributed among activities in order to further maximise this longest path. For fixed choice of δ, it is possible to find an optimal solution to this problem where each α_{ij} is binary. The advantage of binary variables α_{ij} is that products

$\delta_i \alpha_{ij}$ can be easily linearised with the introduction of additional variables. As shown by Bruni et al. (2017), this linearised model is given as follows:

$$\max \sum_{(i,j)\in V^2} \left(\bar{\theta}_i \alpha_{ij} + \hat{\theta}_i w_{ij} - M(1-y_{ij})\alpha_{ij}\right) \tag{5.13}$$

$$\text{s.t.} \sum_{(i,j)\in V^2} \alpha_{ij} - \sum_{(j,i)\in V^2} \alpha_{ji} = 0 \qquad \forall j \in V \tag{5.14}$$

$$\sum_{(0,i)\in V^2} \alpha_{0i} = 1 \tag{5.15}$$

$$\sum_{(i,n+1)\in V^2} \alpha_{i,n+1} = 1 \tag{5.16}$$

$$w_{ij} \le \delta_i \qquad \forall (i,j) \in V^2 \tag{5.17}$$

$$w_{ij} \le \alpha_{ij} \qquad \forall (i,j) \in V^2 \tag{5.18}$$

$$\sum_{i\in V} \delta_i \le \Gamma \tag{5.19}$$

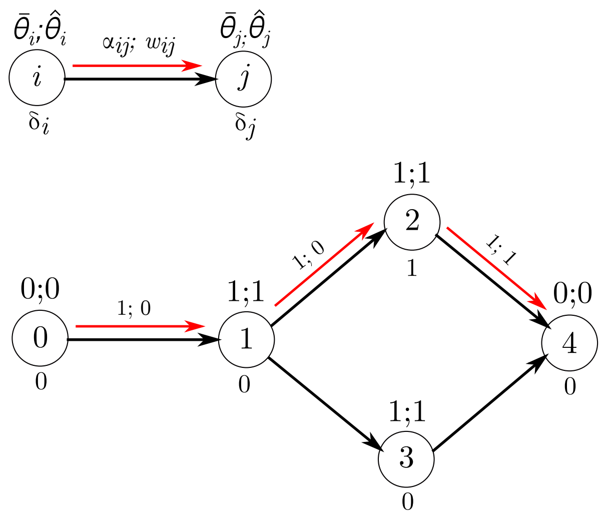
$$0 \le \delta_i \le 1 \qquad \forall i \in V \tag{5.20}$$

$$\alpha_{ij} \in \{0,1\} \qquad \forall (i,j) \in V^2 \tag{5.21}$$

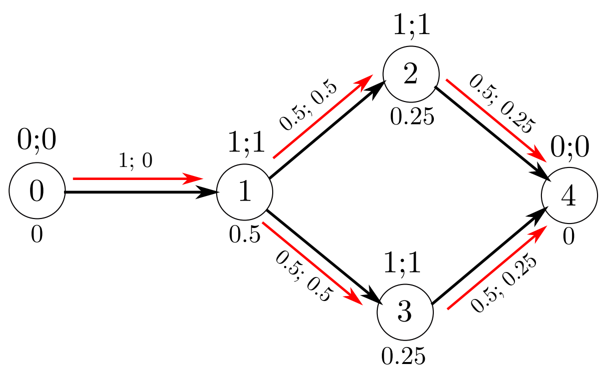
$$w_{ij} \ge 0 \qquad \forall (i,j) \in V^2. \tag{5.22}$$

It is claimed in Proposition 4 of Bruni et al. (2017) that this problem is equivalent to its linear relaxation, where $\alpha_{ij} \in [0,1]$ for all $(i,j) \in V^2$. This, however, is not the case, as the following counter-example demonstrates.

Figure 5.3 shows a project with three non-dummy activities, each with a nominal duration of $\bar{\theta}_i = 1$, and a maximum deviation of $\hat{\theta}_i = 1$, $i = 1,2,3$. Suppose a feasible first-stage solution has been found, resulting in the network shown in Figure 5.3. We consider this problem from the point of view of the adversary, who wishes to distribute up to $\Gamma = 1$ units of delay, in order to maximise the minimum makespan. If (5.13)-(5.22) is equivalent to its linear relaxation, then the adversary gains no advantage by choosing $\alpha \in (0, 1)$ and splitting the unit flow on its route from the source-node 0 to the sink-node 4. However, as can be seen with this example, the adversary does in fact obtain an advantage.



(a) $\alpha_{ij} \in \{0, 1\} \quad \forall (i, j) \in V^2$



(b) $\alpha_{ij} \in [0, 1] \quad \forall (i, j) \in V^2$

Figure 5.3: Counter-example showing that model (5.13)-(5.22) is not equivalent to its linear relaxation.

In Figure 5.3a, $\alpha_{ij} \in \{0,1\}$ for each $(i,j) \in V^2$, and hence the adversary is limited to routing the unit flow through the network via a single path. A worst-case delay in this scenario is that the unit of available delay is entirely assigned to activity 2. Hence, $\delta_2 = 1$, whilst $\delta_1 = \delta_3 = 0$. Minimising the worst-case makespan in this scenario, we get $(\bar{\theta}_1\alpha_{12} + \hat{\theta}_1 w_{12}) + (\bar{\theta}_2\alpha_{24} + \hat{\theta}_2 w_{24}) = (1+0) + (1+1) = 3$.

In Figure 5.3b, $\alpha_{ij} \in [0,1]$ for each $(i,j) \in V^2$, and the adversary is able to split the unit flow into multiple fractional paths on its route through the network. In this case, the adversary can distribute the unit of delay so that $\delta_1 = 0.5$, $\delta_2 = 0.25$, and $\delta_3 = 0.25$. In this scenario, the minimum makespan is $(\bar{\theta}_1\alpha_{12}+\hat{\theta}_1 w_{12})+(\bar{\theta}_1\alpha_{13}+\hat{\theta}_1 w_{13})+(\bar{\theta}_2\alpha_{24}+\hat{\theta}_2 w_{24})+(\bar{\theta}_3\alpha_{34}+\hat{\theta}_3 w_{34}) = (0.5+0.5)+(0.5+0.5)+(0.5+0.25)+(0.5+0.25) = 3.5$, showing that problem (5.13)-(5.22) is not equivalent to its linear relaxation.

Note that Bruni et al. (2017) attempt to prove that model (5.13)-(5.22) is equivalent to its linear relaxation, and therefore polynomially solvable, by showing that the corresponding constraint matrix is totally unimodular. In Appendix B.1, we identify an error with this proof and show that the constraint matrix is not totally unimodular. This result is consistent with the above counter-example.

Since problem (5.13)-(5.22) is not equivalent to its linear relaxation, we cannot apply strong-duality to get an equivalent minimisation problem. Therefore, in order to obtain a compact reformulation of the two-stage robust RCPSP, an alternative reformulation of the adversarial sub-problem is required.

A dynamic programming procedure for solving problem (5.13)-(5.22) when $\Gamma \in \mathbb{Z}$ is presented in Bruni et al. (2017). This procedure works by considering $\Gamma + 1$ paths from the source node 0 to node i, for each $i \in V$, where each path

π_i^γ, $\gamma = 0, \ldots, \Gamma$, is characterised by the inclusion of exactly γ delayed activities. Given a path π_i^γ, its extension to each successor node $j \in Succ_i$ is evaluated by considering two possibilities: either the successor activity j is delayed, resulting in the path $\pi_j^{\gamma+1}$, or it is not delayed, resulting in the path π_j^γ. Hence, the dynamic programming algorithm has a state $ST(j,\gamma)$ for each node j at level γ, and the value of each state $V(ST(j,\gamma))$ is computed through the following recursion:

$$V(ST(0,0)) = 0 \tag{5.23}$$

$$V(ST(j,\gamma)) = \max_{i:(i,j)\in E\cup X} \Big\{ \max \Big(V(ST(i,\gamma)),\, V(ST(i,\gamma-1)) + \bar{\theta}_i + \hat{\theta}_i \Big) \Big\} \qquad \forall j \in V \setminus \{0\},\ \gamma = 1,\ldots,\Gamma \tag{5.24}$$

$$V(ST(j,0)) = \max_{i:(i,j)\in E\cup X} \Big\{ V(ST(i,0)) + \bar{\theta}_i \Big\}. \tag{5.25}$$

This dynamic programming algorithm can be viewed as finding the critical path through the augmented project network built from $\Gamma + 1$ copies of the original project network (an example of such a network is shown in Figure 5.4). The inclusion of an inter-level arc, e.g. a dashed arc in Figure 5.4, in the critical path corresponds to the delay of the activity at the origin of that arc.

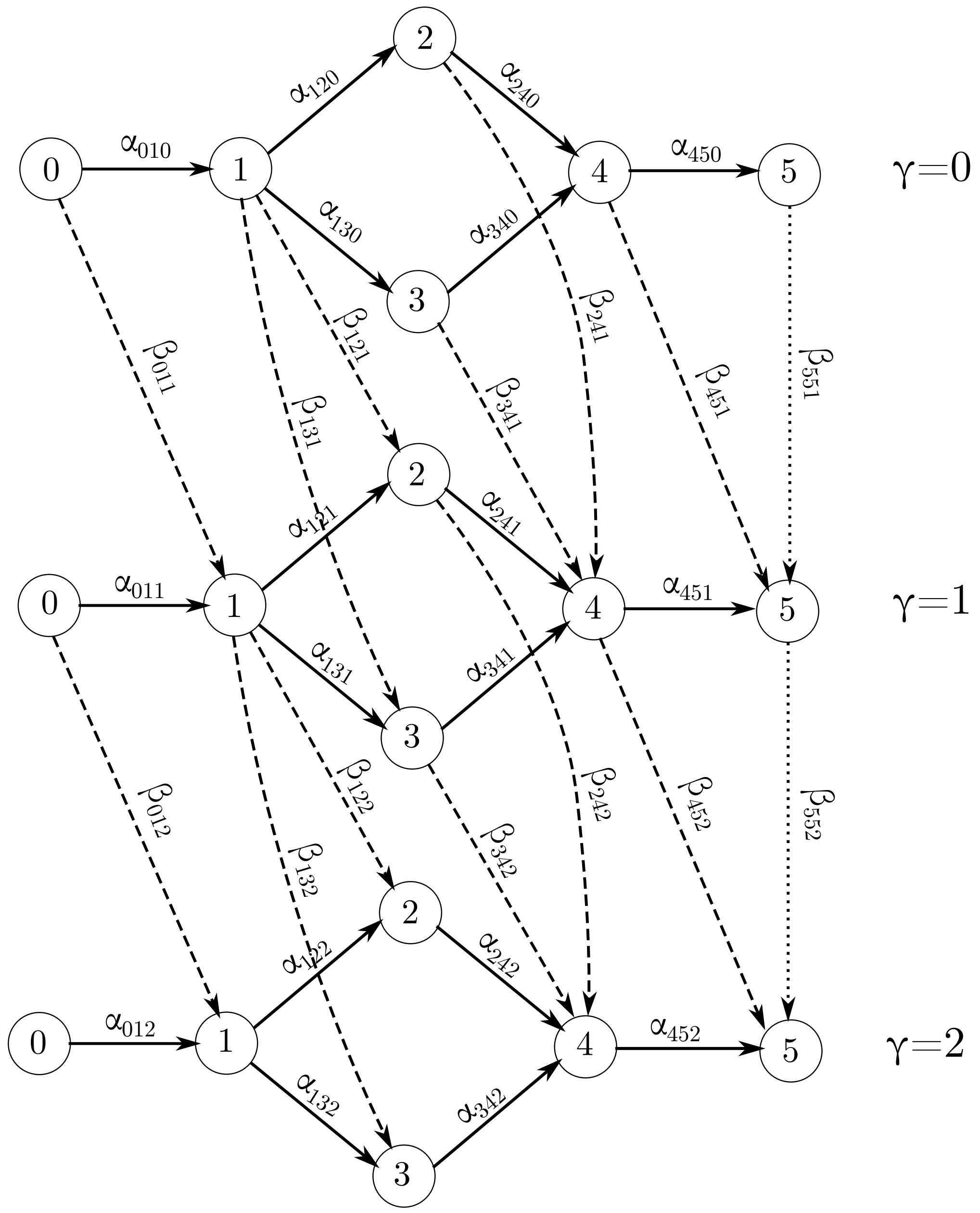


Figure 5.4: Example augmented graph for a project with four non-dummy activities, and where up to $\Gamma = 2$ activities can reach their worst-case durations.

Since the second stage problem is simply a longest-path problem on this augmented network, it can be recast into the following mixed-integer linear program:

$$\max \sum_{(i,j)\in V^2} \sum_{\gamma=0}^{\Gamma} (\bar{\theta}_i - M(1-y_{ij}))\alpha_{ij\gamma} + \sum_{(i,j)\in V^2} \sum_{\gamma=1}^{\Gamma} (\bar{\theta}_i + \hat{\theta}_i - M(1-y_{ij}))\beta_{ij\gamma} \quad (5.26)$$

$$\text{s.t.} \sum_{(j,i)\in V^2} \alpha_{ji\gamma} + \sum_{(j,i)\in V^2} \beta_{ji,\gamma+1} - \sum_{(i,j)\in V^2} \alpha_{ij\gamma} - \sum_{(i,j)\in V^2} \beta_{ij\gamma} = 0 \quad \forall j \in V,\ \gamma = 1,\dots,\Gamma-1 \quad (5.27)$$

$$\sum_{(j,i)\in V^2} \alpha_{ji0} + \sum_{(j,i)\in V^2} \beta_{ji1} - \sum_{(i,j)\in V^2} \alpha_{ij0} = 0 \quad \forall j \in V \quad (5.28)$$

$$\sum_{(j,i)\in V^2} \alpha_{ji\Gamma} - \sum_{(i,j)\in V^2} \alpha_{ij\Gamma} - \sum_{(i,j)\in V^2} \beta_{ij\Gamma} = 0 \quad \forall j \in V \quad (5.29)$$

$$\sum_{(0,i)\in V^2} \alpha_{0i0} + \sum_{(0,i)\in V^2} \beta_{0i1} = 1 \quad (5.30)$$

$$\sum_{(i,n+1)\in V^2} \alpha_{i,n+1,\Gamma} + \sum_{(i,n+1)\in V^2} \beta_{i,n+1,\Gamma} = 1 \quad (5.31)$$

$$\alpha_{ij\gamma} \in \{0,1\} \quad \forall (i,j) \in V^2,\ \gamma = 0,\dots,\Gamma \quad (5.32)$$

$$\beta_{ij\gamma} \in \{0,1\} \quad \forall (i,j) \in V^2,\ \gamma = 1,\dots,\Gamma, \quad (5.33)$$

where $\alpha_{ij\gamma}$ is the flow from node i to node j in level γ and $\beta_{ij\gamma}$ is the flow from node i in level $\gamma-1$ to node j in level γ. The constraints model a unit flow through the augmented network from node 0 in level 0 (Constraint (5.30)) to node $n+1$ in level Γ (Constraint (5.31)). Constraints (5.27) are flow-conservation constraints that ensure that for each node in level $\gamma = 1,\dots,\Gamma-1$, the incoming flow from levels γ and $\gamma-1$ must be equal to the outgoing flow to levels γ and $\gamma+1$. Constraints (5.28) and (5.29) conserve flow over the nodes in the special cases of the first and last level, respectively.

Note that this formulation includes more $\alpha_{ij\gamma}$ and $\beta_{ij\gamma}$ variables than indicated in Figure 5.4, with the edges shown in Figure 5.4 corresponding to the edges for which $y_{ij} = 1$. The edges that are not shown are penalised by constant M in the objective (5.26) when $y_{ij} = 0$. To ensure that it is always possible to find a path from node 0 in level 0 to node $n+1$ in level Γ in the augmented network (if Γ is larger than the number of activities included in the longest path from node 0 to node $n+1$ in the original project network, such a path may not be possible), the final sink nodes of each layer are connected by enforcing $y_{n+1,n+1} = 1$ (see dotted arcs in Figure 5.4). Since $\bar{\theta}_{n+1} + \hat{\theta}_{n+1} = 0$ these additional edges can be traversed at no extra cost to reach node $n+1$ in level Γ.

5.3.2 Compact reformulation

Since the second-stage problem (5.26)-(5.33) is simply a longest-path problem over an augmented project graph, it is equivalent to its linear relaxation where $\alpha_{ij\gamma} \in [0,1]$ for all $(i,j) \in V^2$, $\gamma = 0,\ldots,\Gamma$, and $\beta_{ij\gamma} \in [0,1]$ for all $(i,j) \in V^2$, $\gamma = 1,\ldots,\Gamma$. Hence, we can take the dual of this problem to get an equivalent minimisation problem.

The first-stage problem aims to determine a sufficient selection $X \in \mathcal{X}$ that minimises the second-stage objective value. This first-stage problem can be modelled with a flow-based formulation, as proposed by Artigues et al. (2003). This formulation makes use of continuous resource flow variables f_{ijk}, which determine the amount of resource type $k \in K$ that is transferred upon the completion of activity i to activity j. Additionally, binary variables y_{ij} capture the choice of sufficient selection by representing precedence relationships of the extended project network.

Thus, having dualised the second-stage problem (5.26)-(5.33) into a minimisation problem, the first and second-stages can be combined to obtain the following reformulation of the full two-stage robust RCPSP with budgeted uncertainty:

$$
\begin{aligned}
\min\ & S_{n+1,\Gamma} && && (5.34)\\
\text{s.t. } & S_{00} = 0 && && (5.35)\\
& S_{j\gamma} - S_{i\gamma} \geq \bar{\theta}_i - M(1-y_{ij}) && \forall (i,j) \in V^2,\ \gamma = 0,\dots,\Gamma && (5.36)\\
& S_{j,\gamma+1} - S_{i\gamma} \geq \bar{\theta}_i + \hat{\theta}_i - M(1-y_{ij}) && \forall (i,j) \in V^2,\ \gamma = 0,\dots,\Gamma-1 && (5.37)\\
& y_{ij} = 1 && \forall (i,j) \in E \cup \{(n+1,n+1)\} && (5.38)\\
& f_{ijk} \leq N_k y_{ij} && \forall (i,j) \in V^2,\ \forall k \in K && (5.39)\\
& \sum_{i\in V} f_{ijk} = r_{jk} && \forall j \in V,\ \forall k \in K && (5.40)\\
& \sum_{j\in V} f_{ijk} = r_{ik} && \forall i \in V,\ \forall k \in K && (5.41)\\
& S_{i\gamma} \geq 0 && \forall i \in V,\ \gamma \in 0,\dots,\Gamma && (5.42)\\
& f_{ijk} \geq 0 && \forall (i,j) \in V^2,\ \forall k \in K && (5.43)\\
& y_{ij} \in \{0,1\} && \forall (i,j) \in V^2, && (5.44)
\end{aligned}
$$

where M, as before, is chosen to be greater than or equal to the maximum possible minimum makespan, and N_k is some number greater than or equal to R_k. Constraints (5.35)-(5.37) are the dual constraints of the second-stage problem (5.26)-(5.33), and correspond to makespan constraints that ensure that activity start times respect the project precedence relationships. Constraints (5.38) capture the original project precedences, whilst constraints (5.39)-(5.41) are resource flow constraints. Constraints (5.39) ensure that the resource flows respect the precedence relationships, and constraints (5.40) and (5.41) conserve flow into and out of each node, respectively. Hence problem (5.34)-(5.44) can be seen as a extended makespan minimisation problem, in which a feasible project network must be constructed with the objective of minimising the overall makespan on the corresponding augmented network.

With polynomially many constraints and variables (specifically, $O(|V|^2 \cdot |K| + |V| \cdot \Gamma)$ many variables and $O(|V|^2(|K| + \Gamma) + |E|)$ many constraints), this formulation can be passed directly to standard optimisation software for solving, and the results of doing so are presented in the following section. This is the first formulation of the two-stage adjustable RCPSP for which this is the case. In comparison, the formulation from Bruni et al. (2018) makes use of $O(|V|^2|K| + |V|\Delta)$ many variables and $O(|V|^3 + |V|^2(|K| + \Delta) + |E|)$ many constraints, where $\Delta = \binom{|V|}{\Gamma}$ is an exponential number in Γ.

It is important to note that this basic formulation does not enforce the transitivity of the y-variables. Instead, the formulation captures the extended project network in terms of the y-variables with constraints (5.38) and (5.39), and ensures the feasibility of activity start-times with respect to this extended network through constraints (5.36) and (5.37). In Section 5.4 the computational benefits of extending model (5.34)-(5.44) to include explicit transitivity constraints on the y-variables is examined.

5.4 Computational experiments

This section compares results obtained by solving the compact robust counterpart (5.34)-(5.44), and three slight extensions to this method, with the current state-of-the-art approach to solve the two-stage robust RCPSP proposed in Bruni et al. (2018). Before outlining the proposed extensions to the basic model detailed in the previous section, we introduce the test instances used in this computational study.

The complete sets of results from these experiments as well as Python implementations of each of the four methods we propose can be found at `https://github.com/boldm1/two-stage-robust-RCPSP`.

5.4.1 Instances

The test instances used in this computational study have been converted from deterministic RCPSP instances involving 30 activities, taken from the PSPLIB (Kolisch and Sprecher (1997), `http://www.om-db.wi.tum.de/psplib/`). The difficulty of these instances is measured and controlled by the following three parameters:

1. Network complexity $NC \in \{1.5, 1.8, 2.1\}$. This measures the average number of non-redundant (i.e. non-transitive) arcs per activity.

2. Resource factor $RF \in \{0.25, 0.5, 0.75, 1\}$. This measures the average proportion of resource types for which a non-dummy activity has a non-zero requirement.

3. Resource strength $RS \in \{0.2, 0.5, 0.7, 1\}$. This measures the restrictiveness

of the availability of the resources, with a smaller RS value indicating a more constrained project instance.

The PSPLIB contains a set of 10 instances for each of the 48 possible combinations of instance parameters.

The maximum deviation of the duration of each activity is set to be $\hat{\theta} = \lceil \bar{\theta}/2 \rceil$, where $\bar{\theta}$ is the nominal duration as specified in the original instance file. For each of the 480 deterministic RCPSP instances in the PSPLIB, three robust counterparts have been generated by considering $\Gamma \in \{3, 5, 7\}$, resulting in a total of 1440 test instances. The sets of 30 robust counterparts for each combination of instance parameters are labelled J301, J302, ..., J3048. Note that the instances used in this computational study are identical to the instances used in Bruni et al. (2017) and Bruni et al. (2018).

5.4.2 Implementations

The following section compares the performance of model (5.34)-(5.44) with that of three slight extensions. Here, we outline these extensions and clarify details regarding the practical implementation of these models.

The first variant of the basic model (5.34)-(5.44) includes the following transitivity constraints on the y-variables:

$$y_{ij} + y_{ji} \leq 1 \qquad \forall (i,j) \in V^2 \setminus \{(n+1, n+1)\} \tag{5.45}$$

$$y_{ij} \geq y_{il} + y_{lj} - 1 \qquad \forall (i,l,j) \in V^3. \tag{5.46}$$

As explained in Section 5.3.2, the transitivity of the y-variables is not strictly necessary to ensure the feasibility of the activity start-times. We include them as an extension to model (5.34)-(5.44) in order to assess their impact on the computational performance.

The second extension involves the provision of a heuristic warm-start solution to the solver software. This heuristic solution is obtained with the following procedure:

1. Given an uncertain RCPSP instance, a heuristic solution is found to the corresponding deterministic instance using the latest-finish-time (LFT) priority-rule heuristic (Kolisch, 1996b).

2. From this solution, a feasible set of y-variables is obtained by setting

$$y_{ij} = \begin{cases} 1 & \text{if } s_j \geq f_i \\ 0 & \text{otherwise,} \end{cases}$$

   where s_j is the start time of activity j, and f_i is the finish time of activity i.

3. These y variables are passed to the basic model (5.34)-(5.44), which is solved to provide a feasible warm-start solution.

A detailed example of this warm-start procedure is given in Appendix B.2. This warm-start solution can be used to tighten the big-M constraints (5.36) and (5.37), and thereby further improve the basic model. This is achieved by setting $M_{ij} = LF_i - ES_j$ for each $(i, j) \in V^2$, where ES_j is the earliest start time of activity j, and LF_i is the latest finish time of activity i, calculated relative to the makespan of the warm-start solution. As before, these values are computed recursively via a forward-pass and backward-pass of the project network, respectively.

Note that, although the S-variables of formulation (5.34)-(5.44) are in general continuous, for the purposes of this computational study, the S-variables have been set to be integer. Since $\hat{\theta} = \lceil \bar{\theta}/2 \rceil \in \mathbb{Z}$ for the instances solved in this study, the correctness of the formulation is unaffected by this specification.

In summary, the following section presents results from the following five solution approaches:

1. Basic model (5.34)-(5.44),
2. Basic model with transitivity constraints, i.e (5.34)-(5.46),
3. Basic model with warm-start,
4. Basic model with warm-start and transitivity constraints,
5. Primal method from Bruni et al. (2018). This is the strongest existing approach for solving the two-stage robust RCPSP.

All the models proposed in this chapter have been solved using Gurobi 9.0.1, running on 4 cores of a 2.30GHz Intel Xeon CPU, limited to 16GB RAM. Note that the specifications of this machine have been chosen to be as similar as possible to that of the CPU used in the experiments performed in Bruni et al. (2017) and Bruni et al. (2018). A limit of 20 minutes was imposed on the solution time of each model, the same as used for the experiments performed in Bruni et al. (2017) and Bruni et al. (2018). Results for the primal method have been reproduced from Bruni et al. (2018).

5.4.3 Results

In this section, we first present and analyse results from solving model (5.34)-(5.44) and the three variants proposed in the previous section, before we compare these results with those from the current best iterative algorithm presented in Bruni et al. (2018).

We start by considering the performance profile (Dolan and Moré, 2002) plot shown in Figure 5.5. The performance profile uses the *performance ratio* as a measure by which the different solution methods can be compared. The performance ratio of method $m \in \mathcal{M}$ for problem instance $i \in \mathcal{I}$ is defined to be

$$p_{im} = \frac{t_{im}}{\min_{m \in \mathcal{M}} t_{im}},$$

where t_{im} is the time required to solve instance i using method m. If method m is unable to solve instance i to optimality within the 20 minute time-limit, then $p_{im} = P$, where $P \geq \max_{i,m} t_{im}$. The performance profile of method $m \in \mathcal{M}$ is defined to be the function

$$\rho_m(\tau) = \frac{|\{p_{im} \leq \tau \,:\, i \in \mathcal{I}\}|}{|\mathcal{I}|},$$

i.e. the probability that the performance ratio of method m is within a factor τ of the best performance ratio. The performance profile in Figure 5.5 has been plotted on a log scale for clarity.

It is clear from Figure 5.5 that the provision of a heuristic warm-start solution improves solution time, with the models that make use of a warm-start solution being faster to solve for a greater proportion of instances than their respective

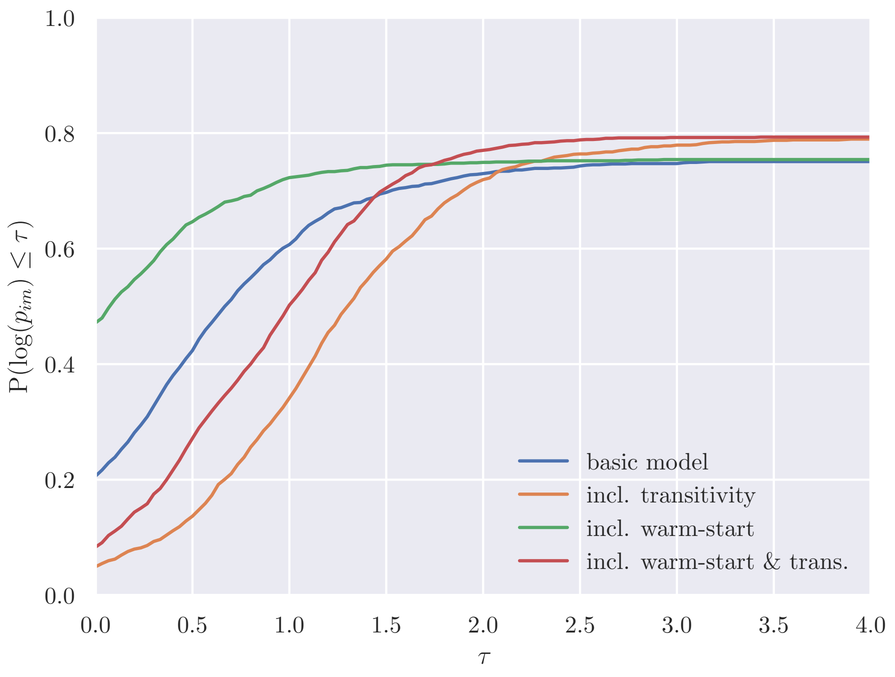


Figure 5.5: Performance profile of relative solution times.

models without a warm-start. It can also be seen that the models that make use of transitivity constraints are slower to solve to optimality for a greater proportion of instances than their respective models that do not use transitivity constraints. However, the inclusion of transitivity constraints does increase the proportion of instances that can be solved to optimality, by 5.3% for the basic model, and by 5.2% for the model with warm-start.

Figure 5.6 plots the cumulative percentage of instances solved to within a given optimality gap within the 20 minute time-limit. Note that the left-hand y-intercept of this figure gives the same information as the right-hand y-intercept in Figure 5.5, that is, the proportion of instances solved to optimality using each method. Looking at Figure 5.6, it can be seen that as well as increasing the proportion of instances that can be solved to optimality, the inclusion of transitivity constraints increases the proportion of instances that can be solved to within a given optimality gap. Of the 255 instances for which an optimal solution was unable to be found with any model, but for which a feasible solution was found using all models, the average optimality gap was 24.53% for the basic model, 22.80% with the inclusion of transitivity constraints, 24.71% with the inclusion of a warm-start solution, and 22.36% with the inclusion of both a warm-start solution and transitivity constraints. Note however that the basic model fails to find a feasible solution for only 3 instances, whilst the model that includes transitivity constraints fails to find a feasible solution for 24 instances. The other two variants find feasible solutions to all 1440 instances.

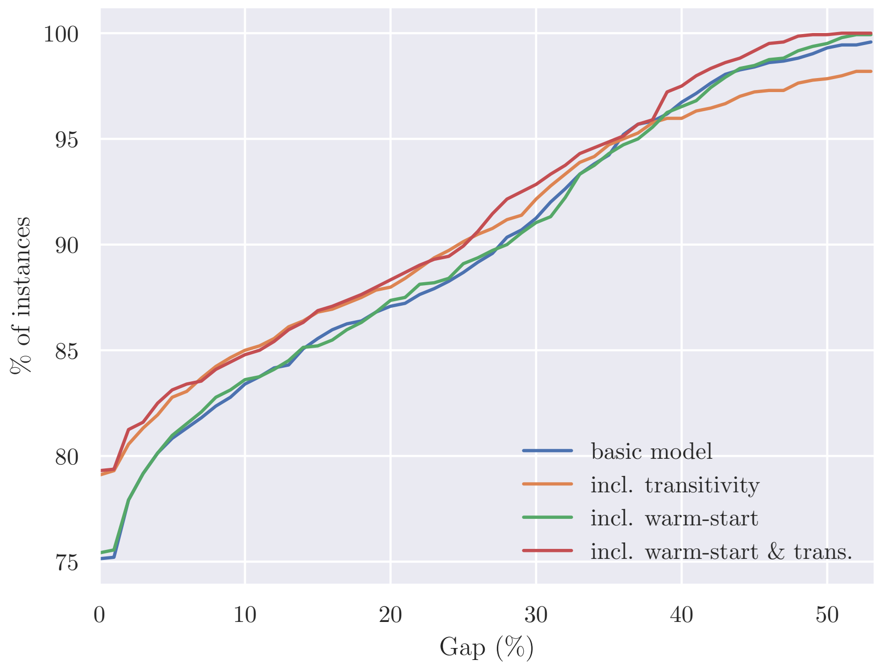


Figure 5.6: Cumulative percentage of instances solved to within given gap of optimality within time-limit.

From Figures 5.5 and 5.6, we can see that the inclusion of a warm-start solution and transitivity constraints in model (5.34)-(5.44) is the best performing variant: it solves the greatest number of instances to optimality, is the strongest

	basic model (5.34)-(5.44)			incl. trans		
Γ	*time*	*gap*	*#solv*	*time*	*gap*	*#solv*
3	318.33	5.18	362	285.28	4.99	388
5	327.37	5.37	361	303.23	5.37	377
7	334.58	5.79	359	308.92	7.05	374
	326.76	5.45	1082	299.14	5.80	1139

	incl. warm-start			incl. warm-start + trans.		
Γ	*time*	*gap*	*#solv*	*time*	*gap*	*#solv*
3	312.77	5.29	366	283.74	4.49	386
5	319.83	5.22	361	292.14	4.60	383
7	329.24	5.49	359	310.07	4.87	373
	320.61	5.33	1086	295.32	4.65	1142

Table 5.1: Comparison of the variants of model (5.34)-(5.44) for different values of Γ.

performing model over the instances which no model can solve to optimality, and is significantly faster to solve than the transitive model without a warm-start.

In Table 5.1, we consider the impact of the uncertainty budget Γ on the performance of the basic model (5.34)-(5.44) and its three variants. For each method we report the average CPU time in seconds (*time*), the average optimality gap in percent (*gap*), and the number of instances solved to optimality (*#solv*). Note that in the case where a method was unable to find a feasible solution to given instance, an optimality gap of 100% has been reported. These results show that although the effect is limited, instances do appear to get more difficult to solve as Γ increases for all four methods.

In Table 5.2, we now compare the performance of the basic model (5.34)-(5.44) and its strongest extension, with the results of the strongest existing algorithm for the two-stage robust RCPSP, the *primal method* (Bruni et al., 2018). For each set of test instances, J301, . . . , J3048, Table 5.2 reports instance parameters (NC, RF, RS), as well as the same measures that were reported in Table 5.1 (*time*, *gap*, *#solv*).

Of the 1440 test instances, 1160 have been solved to optimality within the time-limit by at least one of the four variants of model (5.34)-(5.44) proposed in this chapter. As seen in Table 5.2, the primal method solves 767/1440 instance to optimality, whilst the basic method solves 1082/1440 instances to optimality ($\sim$41% more than the primal method), and the strongest performing method, which includes the warm-start and transitivity constraints, solves 1142/1440 instances to optimality ($\sim$49% more than the primal method). Furthermore, our methods reduce the average gap (4.66% instead of 6.30%) and result in smaller average solution times (295 seconds instead of 621 seconds).

There are only six out of 48 instance sets for which the primal method shows a slightly better performance than the methods we propose (J309, J3013, J3025, J3029, J3041, J3045). These contain some of the most difficult instances, for which all the methods perform poorly. All three methods fail to find an optimal solution to almost all of the instances in these sets, however the primal method achieves a smaller optimality gap in the time limit. The iterative approach utilised by the primal method incrementally improves upon a feasible solution by solving a series of subproblems. For the most challenging instances, this iterative approach is more effective at reducing the optimality gap earlier in the solution process than the methods that we propose, which attempt to solve the full problem at once. It

				primal method (Bruni et al., 2018)			basic model			incl. warm-start + trans.		
	NC	*RF*	*RS*	*time*	*gap*	*#solv*	*time*	*gap*	*#solv*	*time*	*gap*	*#solv*
J301	1.50	0.25	0.20	497.83	1.66	21	6.96	0.00	30	19.69	0.00	30
J302	1.50	0.25	0.50	192.39	0.24	28	2.68	0.00	30	6.57	0.00	30
J303	1.50	0.25	0.70	52.61	0.15	29	1.11	0.00	30	2.42	0.00	30
J304	1.50	0.25	1.00	124.97	1.18	27	0.78	0.00	30	1.62	0.00	30
J305	1.50	0.50	0.20	1200.00	15.92	0	1182.58	16.32	1	1099.26	13.29	8
J306	1.50	0.50	0.50	1115.86	11.56	3	155.59	0.05	29	102.27	0.00	30
J307	1.50	0.50	0.70	605.15	3.01	19	8.10	0.00	30	10.74	0.00	30
J308	1.50	0.50	1.00	363.31	1.85	22	1.45	0.00	30	1.95	0.00	30
J309	1.50	0.75	0.20	1200.00	10.19	0	1200.00	35.81	0	1200.00	30.71	0
J3010	1.50	0.75	0.50	1140.87	20.71	2	791.62	2.04	13	646.61	1.86	20
J3011	1.50	0.75	0.70	974.32	9.84	7	167.99	0.10	28	130.60	0.09	28
J3012	1.50	0.75	1.00	272.53	0.65	26	1.89	0.00	30	2.38	0.00	30
J3013	1.50	1.00	0.20	1200.00	50.55	1	1200.00	40.09	0	1200.00	37.27	0
J3014	1.50	1.00	0.50	1149.39	18.94	2	988.55	4.40	7	853.59	3.44	12
J3015	1.50	1.00	0.70	853.66	5.60	12	129.14	0.37	27	132.07	0.37	27
J3016	1.50	1.00	1.00	207.71	0.74	27	1.33	0.00	30	2.79	0.00	30
J3017	1.80	0.25	0.20	227.35	0.15	28	4.20	0.00	30	7.47	0.00	30
J3018	1.80	0.25	0.50	18.26	0.00	30	1.31	0.00	30	2.20	0.00	30
J3019	1.80	0.25	0.70	65.78	0.35	29	0.80	0.00	30	1.59	0.00	30
J3020	1.80	0.25	1.00	87.68	0.38	28	0.40	0.00	30	1.28	0.00	30
J3021	1.80	0.50	0.20	1200.00	9.28	2	967.73	7.77	10	757.75	5.04	18
J3022	1.80	0.50	0.50	877.51	7.11	10	45.13	0.00	30	43.52	0.00	30
J3023	1.80	0.50	0.70	356.09	0.86	24	2.71	0.00	30	4.65	0.00	30
J3024	1.80	0.50	1.00	201.76	1.13	26	0.95	0.00	30	1.75	0.00	30
J3025	1.80	0.75	0.20	1200.00	13.15	0	1200.00	31.71	0	1200.00	29.77	0
J3026	1.80	0.75	0.50	987.24	6.64	9	271.10	0.39	26	155.52	0.05	29
J3027	1.80	0.75	0.70	628.61	3.51	16	3.29	0.00	30	4.15	0.00	30
J3028	1.80	0.75	1.00	177.53	0.61	27	0.91	0.00	30	1.28	0.00	30
J3029	1.80	1.00	0.20	1200.00	10.5	1	1200.00	42.12	0	1200.00	39.23	0
J3030	1.80	1.00	0.50	1200.00	19.98	0	1158.57	4.51	3	1086.47	3.73	8
J3031	1.80	1.00	0.70	866.16	7.94	9	245.13	0.96	24	236.46	0.63	25
J3032	1.80	1.00	1.00	199.45	1.47	26	1.00	0.00	30	1.16	0.00	30
J3033	2.10	0.25	0.20	28.35	0.00	30	1.58	0.00	30	2.01	0.00	30
J3034	2.10	0.25	0.50	50.02	0.08	29	0.66	0.00	30	0.79	0.00	30
J3035	2.10	0.25	0.70	144.88	1.10	27	0.54	0.00	30	0.65	0.00	30
J3036	2.10	0.25	1.00	20.52	0.00	30	0.29	0.00	30	0.44	0.00	30
J3037	2.10	0.50	0.20	1131.60	5.59	7	634.14	6.69	18	523.56	2.31	23
J3038	2.10	0.50	0.50	463.92	1.59	23	11.63	0.00	30	12.53	0.00	30
J3039	2.10	0.50	0.70	268.81	0.78	27	3.55	0.00	30	2.56	0.00	30
J3040	2.10	0.50	1.00	257.42	1.66	24	1.29	0.00	30	1.14	0.00	30
J3041	2.10	0.75	0.20	1200.0	7.12	1	1200.00	26.73	0	1193.00	20.67	1
J3042	2.10	0.75	0.50	886.16	7.42	10	282.54	1.43	26	169.89	0.38	27
J3043	2.10	0.75	0.70	823.56	4.69	12	171.58	0.13	27	30.39	0.00	30
J3044	2.10	0.75	1.00	481.39	3.38	19	1.56	0.00	30	1.44	0.00	30
J3045	2.10	1.00	0.20	1164.00	8.10	2	1200.00	34.92	0	1200.00	31.91	0
J3046	2.10	1.00	0.50	1200.00	16.45	0	971.15	4.47	7	836.19	2.79	16
J3047	2.10	1.00	0.70	866.38	7.79	9	259.45	0.35	26	80.91	0.00	30
J3048	2.10	1.00	1.00	181.33	0.83	26	1.33	0.00	30	1.59	0.00	30
				621.09	6.30	767	326.76	5.24	1082	295.32	4.66	1142

Table 5.2: Comparison of primal method (Bruni et al., 2018), basic model (5.34)-(5.44), and extended model including warm-start and transitivity constraints.

is important to note however that this does not necessarily mean that the primal method is able to solve these instances more quickly than the methods we propose, and that the overall solution times of all three methods on these instances remain unknown. Note also that the primal method solves some instances to optimality whilst simultaneously reaching the maximum time-limit of 1200 seconds. It is unclear whether or not this is a numerical inaccuracy in the results presented in Bruni et al. (2018).

Overall, we find that the methods proposed in this chapter considerably outperform the previous best approach, solving almost 50% more instances in a considerably shorter computation time. While the primal method has the drawback that several models have to be solved subsequently in an iterative process, our reformulation makes it possible to solve the two-stage adjustable RCPSP with a single mixed-integer program, utilising the strength of current solvers such as Gurobi.

5.5 Conclusion

This chapter has introduced a new mixed-integer linear programming formulation for the robust counterpart to the two-stage adjustable robust RCPSP. This new compact formulation has been derived by considering a reformulation of the second-stage adversarial sub-problem of maximising the worst-case delayed makespan for a project without resource conflicts. The reformulation of this sub-problem is equivalent to a longest-path problem over an augmented project network made from multiple copies of the original project network. Hence, the dual of this longest-path problem can be inserted into the first-stage resource allocation problem to obtain a compact minimisation problem for the full two-stage robust RCPSP.

The performance of this new formulation has been examined over 1440 instances of varying characteristics and difficulty. Results show that the proposed formulation can be solved by standard optimisation software significantly faster than the current best algorithm for solving this problem. Using our approach, almost 50% more instances can be solved to optimality within the same time-limit, while also achieving a smaller average gap and a smaller average solution time.

Regarding future research on the two-stage robust RCPSP, the development of heuristic approaches for solving larger and more-challenging instances of this problem would seem to be a natural and worthwhile objective.

Chapter 6

A Faster Exact Method for Solving the Robust Multi-Mode Resource-Constrained Project Scheduling Problem

6.1 Introduction

The multi-mode resource-constrained project scheduling problem (MRCPSP) is a generalisation of the widely-studied resource-constrained project scheduling problem (RCPSP) to include multiple processing modes for each activity. The inclusion of these modes allows the modelling of situations in which there is more than one way of executing project activities, with each option having its own duration and resource requirements. The MRCPSP consists of selecting the processing modes and start times for a given set of activities, subject to a set of precedence constraints and limited resource availability, with the objective of minimising the overall project duration, known as the makespan.

In this chapter we consider the MRCPSP under uncertain activity durations and model it using a two-stage adjustable robust optimisation framework. In this setting, a first-stage problem is solved to determine activity mode selections and make activity sequencing decisions to resolve resource conflicts. Following this, the actual activity durations are realised and a complete schedule is computed. The aim of the two-stage adjustable robust MRCPSP is to find a feasible first-stage solution (i.e. mode selection and sequencing decisions) in order to minimise the realised worse-case makespan, as computed in the second-stage. We refer to this problem as the robust MRCPSP.

A number of papers in recent years have applied this two-stage robust optimisation approach to the RCPSP. First to use this approach were Artigues et al. (2013b), who presented an iterative scenario-relaxation algorithm for this problem with the objective of minimising the worst-case absolute regret. More recently, Bruni et al. (2017) introduced a Benders'-style decomposition approach for solving the robust RCPSP with the objective of minimising the worst-case project makespan. This work was extended in Bruni et al. (2018), which presented a computational study comparing an additional Benders' decomposition approach against a primal decomposition algorithm. Most recently, the work presented in the previous chapter (Bold and Goerigk, 2021) introduced a compact reformulation of the robust RCPSP and presented results which showed the superior computational performance of that formulation over the iterative decomposition-based methods developed in the two preceding papers.

The application of this two-stage robust optimisation approach for the MRCPSP is a very recent development. To the best of our knowledge, the only existing paper to consider this problem is Balouka and Cohen (2021), in which the Benders' decomposition approach introduced by Bruni et al. (2017) for the robust RCPSP has been extended for application to the MRCPSP. Mirroring that extension, this chapter adapts the compact formulation developed in the previous chapter (Bold and Goerigk, 2021) to the MRCPSP, with the aim of achieving similarly superior computational performance over the Benders' solution approach.

Following a formal description of the two-stage robust MRCPSP in Section 6.2, we outline the proposed compact formulation for this problem in Section 6.3 and present a strengthened version of the Benders' decomposition solution approach from Balouka and Cohen (2021) in Section 6.4. Results from a computational comparison of these two approaches are detailed in Section 6.5.

6.2 Problem description

A project consists of a set of non-preemptive activities $V = \{0, 1, \dots, n, n+1\}$, where 0 and $n+1$ denote the dummy source and sink activities respectively. Each activity $i \in V$ has a set of available processing modes given by $M_i = \{1, \dots, |M_i|\}$. The nominal duration of activity i when executed in mode $m \in M_i$ is given by $\bar{d}_{im}$, whilst its worst-case duration is given by $\bar{d}_{im} + \hat{d}_{im}$, where $\hat{d}_{im}$ is its maximum durational deviation. Each mode for activity i, $m \in M_i$, has an associated renewable resource requirement of r_{imk} for each $k \in K$, where K is the set of renewable resource types involved in the project. Each renewable resource $k \in K$ has an availability of R_k at each time period in the project horizon. As well as renewable resource requirements, mode m of activity i also has a non-renewable resource requirement of $r^{'}_{imk}$ for each non-renewable resource type $k \in K^{'}$, with each non-renewable resource having an overall availability of $R^{'}_k$ for the entire project horizon. Additionally, the project is subject to a set of strict finish-to-start precedence constraints given by E, where $(i, j) \in E$ enforces that activity i must finish before activity j can begin. These form a project network that can be represented using a directed graph $G(V, E)$. Figure 6.1 shows an example instance involving five non-dummy activities and a single renewable resource.

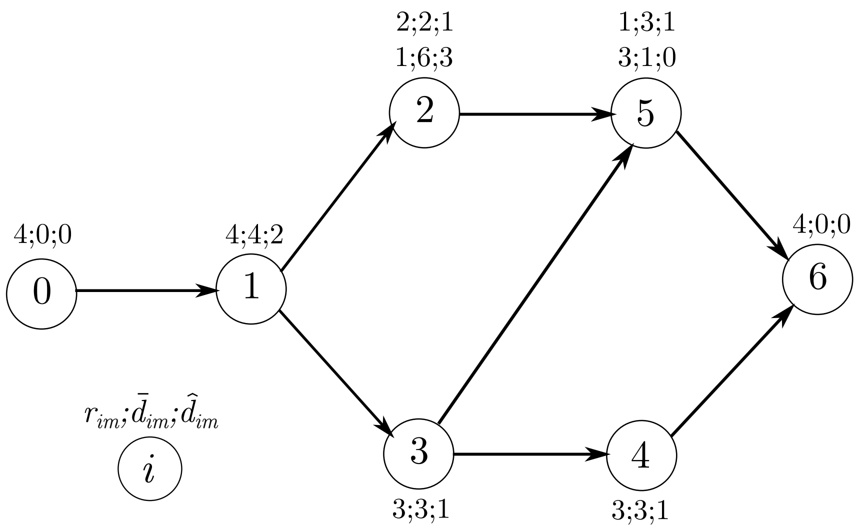


Figure 6.1: An example instance involving five non-dummy activities and a single renewable resource with availability $R_1 = 4$. Activities 2 and 5 each have two available processing modes, whilst the other activities have only a single mode.

For a given choice of processing modes for each activity $\boldsymbol{m} = (m_1, \ldots, m_n)$, we assume that the activity durations lie somewhere in a budgeted uncertainty set of the form

$$\mathcal{U}_{\boldsymbol{m}}(\Gamma) = \left\{ \boldsymbol{d_m} \in \mathbb{R}_+^{|V|} : d_{im_i} = \bar{d}_{im_i} + \xi_i \hat{d}_{im_i},\ 0 \leq \xi_i \leq 1,\ \forall i \in V,\ \sum_{i \in V} \xi_i \leq \Gamma \right\}.$$

Introduced by Bertsimas and Sim (2004), the motivation of the budgeted uncertainty set is to control the pessimism of the solution by introducing a robustness parameter Γ to limit the number of jobs that can simultaneously achieve their worst-case durations. Observe that when $\Gamma = 0$, each activity takes its nominal duration and the resulting problem is the deterministic MRCPSP. At the other extreme, when $\Gamma = n$, every activity takes its worst-case duration, in which case the uncertainty set becomes equivalent to an interval uncertainty set. In this case the problem can again be solved as a deterministic MRCPSP instance considering only worst-case durations.

For a given set of activity modes $\boldsymbol{m}$, a forbidden set is defined to be any subset of activities $F_{\boldsymbol{m}} \subseteq V$ that are not precedence-related, such that $\sum_{i \in F_{\boldsymbol{m}}} r_{im_ik} > R_k$ for at least one resource $k \in K$. That is, a forbidden set is a collection of activities that cannot be executed in parallel only because of resource limitations. Applying the main representation theorem of Bartusch et al. (1988), for a particular choice of activity modes $\boldsymbol{m}$, a solution to the MRCPSP can be defined by a set of additional precedences $X_{\boldsymbol{m}} \subseteq V^2 \backslash E$ such that the extended precedence network $G(V, E \cup X_{\boldsymbol{m}})$ is acyclic and contains no forbidden sets. Such an extension to the project network is referred to as a *sufficient selection*.

The aim of the two-stage robust MRCPSP is therefore to determine activity modes and a corresponding sufficient selection in order to minimise the worst-case project makespan. For the case where activity durations lie in a budgeted uncertainty set that we consider in this chapter, this problem can be written as

$$\min_{\boldsymbol{m} \in \mathcal{M},\, X_{\boldsymbol{m}} \in \mathcal{X}_{\boldsymbol{m}}} \max_{\boldsymbol{d} \in \mathcal{U}_{\boldsymbol{m}}(\Gamma)} \min S_{n+1} \tag{6.1}$$

$$\text{s.t. } S_0 = 0 \tag{6.2}$$

$$S_j - S_i \geq d_{im_i} \qquad \forall (i,j) \in E \cup X_{\boldsymbol{m}} \tag{6.3}$$

$$S_i \geq 0 \qquad \forall i \in V, \tag{6.4}$$

where $\mathcal{M} \subseteq \mathbb{N}^n$ represents the set of all possible combinations of activity processing mode selections, and $\mathcal{X}_{\boldsymbol{m}}$ is the set of all possible sufficient selections for the choice of processing modes given by $\boldsymbol{m}$.

An optimal solution to the instance shown in Figure 6.1 is given by Figure 6.2, where the mode choices are shown in bold, and the sufficient selection is given by $\{(3,2)\}$. The worst-case schedule for this optimal solution is shown in Figure 6.3, where activities 1 and 3 have been delayed to achieve a worst-case makespan of 15 (note that delaying a combination of activity 1 and any other activity results in the same worst-case makespan).

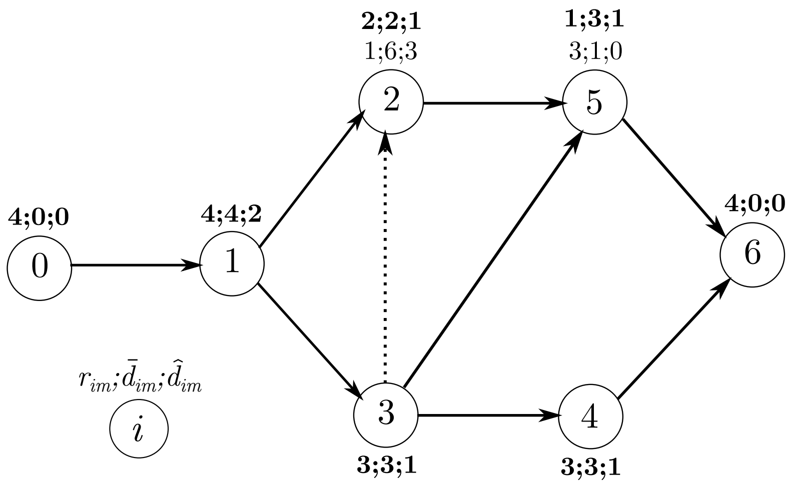


Figure 6.2: Optimal solution to the example instance in Figure 6.1. Activity mode choices are highlighted in bold and the sufficient selection is given by dashed arc $\{(3,2)\}$.

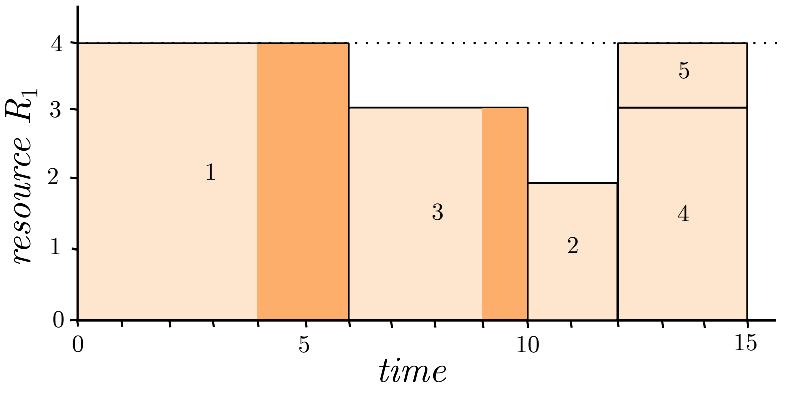


Figure 6.3: Worst-case schedule corresponding to optimal robust solution in Figure 6.2, where activities 1 and 3 have been delayed.

6.3 A compact formulation

We propose solving the robust MRCPSP using an extended version of the mixed-integer programming formulation developed by Bold and Goerigk (2021) for solving the two-stage adjustable robust RCPSP. This formulation combines the first and second-stage problems into a single compact formulation and was shown to significantly outperform the strongest iterative decomposition-based methods for that problem.

This formulation is constructed by recasting the adversarial subproblem of determining the worst-case activity durations for a given set of mode choices and sufficient selection as a longest path problem on an augmented project network. This directed acyclic graph is formed of $\Gamma + 1$ connected copies of the original project network, with each level being used to account for a delay to a single activity in the project (a similar construction has also been applied in Bendotti et al. (2019)). Having reformulated the adversarial subproblem as a longest-path problem, strong duality can be employed to enable its insertion into a formulation for the first-stage problem, and ultimately arrive at a complete compact formulation for the full problem. Given that the derivation of this formulation is more or less identical to the derivation of the corresponding formulation for the robust RCPSP, we omit it here and instead refer the reader to Bold and Goerigk (2021).

In the resulting compact formulation, y_{ij} variables are used to define a complete set of transitive precedences between the project activities that are formed by the original project precedence constraints and the additional precedences introduced to resolve resource conflicts, i.e. $\{(i,j),\, i,j \in V : y_{ij} = 1\} = T(E \cup X_{\boldsymbol{m}})$, where $T(\cdot)$ is used to denote the transitive closure of a set. Variables x_{im} determine the processing mode for each activity, whilst resource flow variables f_{ijk} track the amount of renewable resource k that is transferred from activity i to activity j upon its completion. The $S_{i\gamma}$ variables are dual variables associated with the longest-path subproblem, which in this formulation are used to track the start time of activities under the worst-case activity durations. Using these variables, the compact formulation for the robust MRCPSP can be written as

$$\min\ S_{n+1,\Gamma} \tag{6.5}$$

$$\text{s.t.}\ \ S_{00} = 0 \tag{6.6}$$

$$S_{j\gamma} - S_{i\gamma} \geq \bar{d}_{im} x_{im} - N(1 - y_{ij}) \qquad \forall (i,j) \in V^2,\ \forall m \in M_i,\ \gamma = 0, \dots, \Gamma \tag{6.7}$$

$$S_{j,\gamma+1} - S_{i\gamma} \geq (\bar{d}_{im} + \hat{d}_{im}) x_{im} - N(1 - y_{ij}) \qquad \forall (i,j) \in V^2,\ m \in M_i,\ \gamma = 0, \dots, \Gamma - 1 \tag{6.8}$$

$$y_{ij} = 1 \qquad \forall (i,j) \in E \cup \{(n+1, n+1)\} \tag{6.9}$$

$$y_{ij} + y_{ji} \leq 1 \qquad \forall i,j \in V,\ i < j \tag{6.10}$$

$$y_{ij} = y_{ip} + y_{pj} - 1 \qquad \forall i, j, p \in V,\ i \neq j \neq p \tag{6.11}$$

$$f_{ijk} \leq P_{ijk} y_{ij} \qquad \forall (i,j) \in V^2,\ i \neq n+1,\ j \neq 0,\ \forall k \in K \tag{6.12}$$

$$\sum_{i \in V \setminus \{n+1\}} f_{ijk} = \sum_{m \in M_j} r_{jmk} x_{jm} \qquad \forall j \in V \setminus \{0\},\ \forall k \in K \tag{6.13}$$

$$\sum_{j \in V \setminus \{0\}} f_{ijk} = \sum_{m \in M_i} r_{imk} x_{im} \qquad \forall i \in V \setminus \{n+1\},\ \forall k \in K \tag{6.14}$$

$$\sum_{m \in M_i} x_{im} = 1 \qquad \forall i \in V \tag{6.15}$$

$$\sum_{m \in M_i} r'_{imk} x_{im} \leq R'_k \qquad \forall i \in V,\ k \in K' \tag{6.16}$$

$$S_{i\gamma} \geq 0 \qquad \forall i \in V,\ \gamma \in 0, \dots, \Gamma \tag{6.17}$$

$$y_{ij} \in \{0,1\} \qquad \forall (i,j) \in V^2 \tag{6.18}$$

$$f_{ijk} \geq 0 \qquad \forall (i,j) \in V^2,\ \forall k \in K \tag{6.19}$$

$$x_{im} \in \{0,1\} \qquad \forall i \in V,\ m \in M_i, \tag{6.20}$$

where $N = \sum_{i \in V} \max_{m \in M_i} (\bar{d}_{im} + \hat{d}_{im})$ is an upper bound on the minimum makespan, and $P_{ijk} = \min\{\max_{m \in M_i} r_{imk},\ \max_{m \in M_j} r_{jmk}\}$ is the maximum possible flow of resource k from i into j.

Constraints (6.6)-(6.8) are the dual constraints corresponding to the longest-path variables in the adversarial subproblem, and serve to ensure that the activity start times respect the project precedences as well as any delays to activity durations. Constraints (6.9) capture the original project precedence relationships. Although not required for the correctness of the model, constraints (6.10) and (6.11) are additional transitivity constraints that have been included because they were shown in Bold and Goerigk (2021) to provide significant improvements to the computational performance of the model. Constraints (6.12)-(6.14) are resource flow constraints which ensure that the transfer of resources between activities follows precedence constraints and that resources are conserved as they flow into and out of each activity in the network. Constraints (6.15) allows only one processing mode to be selected for each activity. Finally, non-renewable resource constraints are enforced by (6.16).

With polynomially many variables and constraints this formulation can be implemented straightforwardly using standard mathematical optimisation software such as Gurobi or CPLEX.

6.4 A Benders' decomposition approach

In this section we outline an alternative approach to solving the robust MRCPSP based on Benders' decomposition. A Benders'-type approach for solving the robust MRCPSP was first presented in Balouka and Cohen (2021), and as mentioned previously, this itself was based on the approach for solving the robust RCPSP used in Bruni et al. (2017).

The main idea behind this approach is to decompose the full problem into its two stages: 1. a master problem that determines activity processing modes and resolves resource conflicts, and 2. a subproblem that takes the solution from the master problem and evaluates it by finding the worst-case makespan for the resulting network by solving a longest path problem. Since the master problem does not account for all the problem uncertainty at once, its solution forms a lower bound to the optimal objective value of the original problem. Meanwhile, the solution to the subproblem is feasible to the original problem, and therefore provides an upper bound. These two problems are solved iteratively, with optimality cuts being added to the master problem at each iteration based on the solution to the subproblem until the lower and upper bounds meet.

After initially implementing the Benders' algorithm as it is presented in Balouka and Cohen (2021), it was discovered that its performance could be strengthened by replacing the master problem formulation used in that implementation with a stripped-down version of the compact formulation (6.5)-(6.20) in which the uncertainty is removed. In this section we present our strengthened implementation of the Benders' approach for solving the robust MRCPSP. A comparison of these two Benders' implementations is included in the results in Section 6.5. For the details of the original implementation of the Benders' decomposition algorithm for the robust MRCPSP, see Balouka and Cohen (2021).

6.4.1 The master problem

The role of the master problem is to determine a choice of activity processing modes $\boldsymbol{m}$, as well as a sufficient selection $X_{\boldsymbol{m}}$ to resolve any resulting resource conflicts. This problem is solved without the direct consideration of the uncertain activity durations, and instead, uncertainty is accounted for in the subproblem and communicated back to the master problem with the use of optimality cuts. Hence, to remove the uncertainty from the model, (6.5)-(6.20) is modified by considering only a single level of the augmented project network. Using this formulation, the master problem at iteration t can be written as

$$\min \eta \tag{6.21}$$

$$\text{s.t. } \eta \geq S_{n+1} \tag{6.22}$$

$$S_0 = 0 \tag{6.23}$$

$$S_j - S_i \geq \bar{d}_{im} x_{im} - N(1 - y_{ij}) \qquad \forall (i,j) \in V^2, \forall m \in M_i \tag{6.24}$$

$$\eta \geq \lambda(\boldsymbol{x}, \boldsymbol{y}, V^{*\ell}) \qquad \forall \ell = 1, \ldots, t-1 \tag{6.25}$$

$$(6.9)\text{-}(6.16), (6.18)\text{-}(6.20) \tag{6.26}$$

$$S_i \geq 0 \qquad \forall i \in V, \tag{6.27}$$

where (6.25) are the optimality cuts generated from the solutions of the resulting subproblems at each of the previous iterations.

The optimal objective value of the master problem (6.21)-(6.27), denoted by η^*, forms a lower bound to the original robust MRCPSP problem. Note that for the first iteration, the solution to the master problem corresponds to an optimal solution to the MRCPSP with nominal activity durations.

6.4.2 The subproblem

The subproblem at iteration t computes the worst-case makespan for the mode choices and sufficient selection from the master problem at iteration t, denoted by $\boldsymbol{m}^{*t}$ and $X_{\boldsymbol{m}^{*t}}$ respectively. This is done by solving a longest-path problem through the extended project network $T(E \cup X_{\boldsymbol{m}^{*t}})$, in which the activity durations (i.e. arc lengths) can be chosen from the budgeted uncertainty set $\mathcal{U}_{\boldsymbol{m}^{*t}}(\Gamma)$. This problem can be formulated as

$$V^{*t} = \max \sum_{(i,j) \in T(E \cup X_{\boldsymbol{m}^{*t}})} \bar{d}_{im_i^{*t}} \alpha_{ij} + \hat{d}_{im_i^{*t}} w_{ij} \tag{6.28}$$

$$\sum_{(i,n+1) \in T(E \cup X_{\boldsymbol{m}^{*t}})} \alpha_{i,n+1} = 1 \tag{6.29}$$

$$\sum_{(0,i)\in T(E\cup X_{\boldsymbol{m}^{*t}})} \alpha_{0,i} = 1 \tag{6.30}$$

$$\sum_{(i,j)\in T(E\cup X_{\boldsymbol{m}^{*t}})} \alpha_{ij} - \sum_{(j,i)\in T(E\cup X_{\boldsymbol{m}^{*t}})} \alpha_{ji} = 0 \qquad \forall i \in V \setminus \{0, n+1\} \tag{6.31}$$

$$w_{ij} \leq \xi_i \qquad \forall (i,j) \in T(E \cup X_{\boldsymbol{m}^{*t}}) \tag{6.32}$$

$$w_{ij} \leq \alpha_{ij} \qquad \forall (i,j) \in T(E \cup X_{\boldsymbol{m}^{*t}}) \tag{6.33}$$

$$\sum_{i\in V} \xi_i \leq \Gamma \tag{6.34}$$

$$\alpha_{ij} \in \{0,1\} \qquad \forall (i,j) \in T(E \cup X_{\boldsymbol{m}^{*t}}) \tag{6.35}$$

$$w_{ij} \geq 0 \qquad \forall (i,j) \in T(E \cup X_{\boldsymbol{m}^{*t}}) \tag{6.36}$$

$$0 \leq \xi_i \leq 1 \qquad \forall i \in V. \tag{6.37}$$

Variables α_{ij} define the longest path through the network, which we denote as $\pi^{*t} = \{(i,j),\ i,j \in V : \alpha_{ij} = 1\}$, and has length V^{*t}. Variables ξ_i are the activity delay variables, whilst w_{ij} are used to linearise the formulation. Note that this formulation of the subproblem is identical to the one implemented by Balouka and Cohen (2021).

6.4.3 Optimality cuts

Following the solution to the subproblem, a valid cut is generated and added to the master problem for the next iteration. This cut simply forces an alternative solution in the master problem if it is to achieve a better objective value in the next iteration.

Given the current best lower bound for the original problem, LB, the mode selection from the master problem, $\boldsymbol{m}^{*t}$, and the optimal objective value of the subproblem, V^{*t}, the cut at iteration t can be written as

$$\eta \geq (V^{*t} - LB) \cdot \sum_{(i,j)\in\pi^{*t}} \Big(1/3(y_{ij} + x_{i,m_i^{*t}} + x_{j,m_j^{*t}}) - (3 - y_{ij} - x_{i,m_i^{*t}} - x_{j,m_j^{*t}})\Big) - (V^{*t} - LB) \cdot (|\pi^{*t}| - 1) + LB. \quad (6.38)$$

Balouka and Cohen (2021) show that this constraint is a valid optimality cut for the problem, and that the number of these cuts that need to be added to the master problem before finding an optimal solution is finite.

An overview of the implementation of the Benders' solution approach outlined in this section is presented in Algorithm 6.1.

Algorithm 6.1 Benders' decomposition algorithm.

```
1:  **Initialise:** Set $LB = -\infty$, $UB = +\infty$ and $t = 1$.
2:  **while** $UB > LB$ **do**
3:      **Solve master problem (6.21)-(6.27)**
4:          Get objective value $\eta^{*t}$, processing modes $\boldsymbol{m}^{*t}$ and precedences $\boldsymbol{y}_{\boldsymbol{m}^{*t}}$
5:          If $\eta^{*t} > LB$, update $LB \leftarrow \eta^{*t}$
6:      **Solve subproblem (6.28)-(6.37)**
7:          Get objective value $V^{*t}$ and longest path $\pi^{*t}$
8:          If $V^{*t} < UB$, update $UB \leftarrow V^{*t}$
9:      **Add cut (6.38) to master problem**
10:     Update $t \leftarrow t + 1$
11: **end while**
12: **return** $UB$
```

6.4.4 Example

To demonstrate its implementation, we use the Benders' decomposition approach to solve the example shown in Figure 6.1 with $R_1 = 4$ and $\Gamma = 2$. The algorithm solves this instance in 6 iterations, and the solution information from the master and subproblem at each of these iterations is shown in Table 6.1.

The first iteration of the master problem solves the nominal instance, assuming no delays to the activity durations. This solution opts to schedule activities 2 and 3 in parallel, which can be achieved by setting $m_2 = 2$, and the sufficient selection is given by $\{(4, 5)\}$. This first solution is evaluated in the subproblem to have a worst-case makespan of 16. In the second iteration, having added the first optimality cut, the master problem finds the solution shown in Figure 6.2. By calculating the schedule shown in Figure 6.3, the subproblem evaluates the objective value of this solution to be 15. Although this solution is optimal, the algorithm requires a further four iterations to prove that this is the case. Note that there is no need to solve the subproblem in the final iteration since after solving the master problem and updating UB, we have that $LB = UB$.

t	LB	UB	η^{*t}	V^{*t}	$\boldsymbol{m}^{*t}$	π^{*t}
1	11	16	11	16	1,2,1,1,2	0,1,2,5,6
2	12	15	12	15	1,1,1,1,1	0,1,3,2,4,6
3	12	15	12	15	1,1,1,1,1	0,1,2,3,4,6
4	13	15	13	18	1,2,1,1,1	0,1,2,5,6
5	13	15	13	16	1,1,1,2,1	0,1,3,4,2,5,6
6	15	15	15	-	1,1,1,1,1	-

Table 6.1: Solution information at each of the six iterations of the Benders' algorithm required to solve example instance in Figure 6.1.

6.5 Computational experiments and results

In this section we compare results from using the compact formulation and the Benders' decomposition approach to solve uncertain MRCPSP instances. A complete set of the raw results used to generate the tables and plots presented here, in addition to the source code used to implement these experiments, can be found at `https://github.com/boldm1/robust-mrcpsp`.

The instances used in this computational study have been created from the deterministic $j10$, and $j20$ MRCPSP instances from the PSPLIB (Kolisch and Sprecher (1997), `https://www.om-db.wi.tum.de/psplib/`). Note that these are the same instance sets as used for the experiments in Balouka and Cohen (2021). The $j10$ set contains a total of 536 instances each involving 10 activities, and the $j20$ set contains a total of 554 instances each involving 20 activities. We introduce uncertainty into these deterministic instances by setting the maximum durational deviation for each activity to be $\hat{d}_{im} = \lfloor 0.7 \times \bar{d}_{im} \rfloor$ for each mode $m \in M_i$. These uncertain instances have then been solved using both methods for a range of robustness levels Γ. In particular we solve the $j10$ instances for $\Gamma \in \{0, 3, 5, 7\}$ and the $j20$ instances for $\Gamma \in \{0, 5, 10, 15\}$.

Both the compact reformulation and Benders' decomposition approach have been implemented in Python 3.9.2 and solved using Gurobi 9.0.1 running on a single core of a 2.30 GHz Intel Xeon CPU. A time limit of 2 hours per instance was imposed on each of the solution methods.

We begin by observing the effect of the robustness parameter Γ on the average optimal objective values shown in Tables 6.2a and 6.2b. The values in these tables have been computed only over the instances for which an optimal solution was found for all the values of Γ (i.e. all 536 instances in the $j10$ set, and 477 instances of the $j20$ set). As we would expect, the solution cost increases in a concave manner as Γ increases.

Γ	0	3	5	7
obj.	16.84	25.34	26.35	26.46

(a) $j10$

Γ	0	5	10	15
obj.	24.61	37.89	38.69	38.69

(b) $j20$

Table 6.2: Average optimal objective values across instances in the $j10$ and $j20$ instance sets for different values of Γ.

Table 6.3 reports the percentage of instances solved to optimality (*%sol.*), the average percentage optimality gap over instances for which a feasible solution was found (*gap*), and the average solution time in seconds (*sol. time*) for the different choices of Γ across the two instance sets, for both the Benders' decomposition approach and the compact formulation. Note that if no optimal solution was found within the time limit by one of the solution methods, the solution time was recorded as the time limit value of 7200 seconds. Additionally, for the Benders' approach, the average number of (completed) iterations (*it.*) and the average time per (completed) iteration (*it. time*) are also reported. The average of the reported values across the different values for Γ are also given for each instance set.

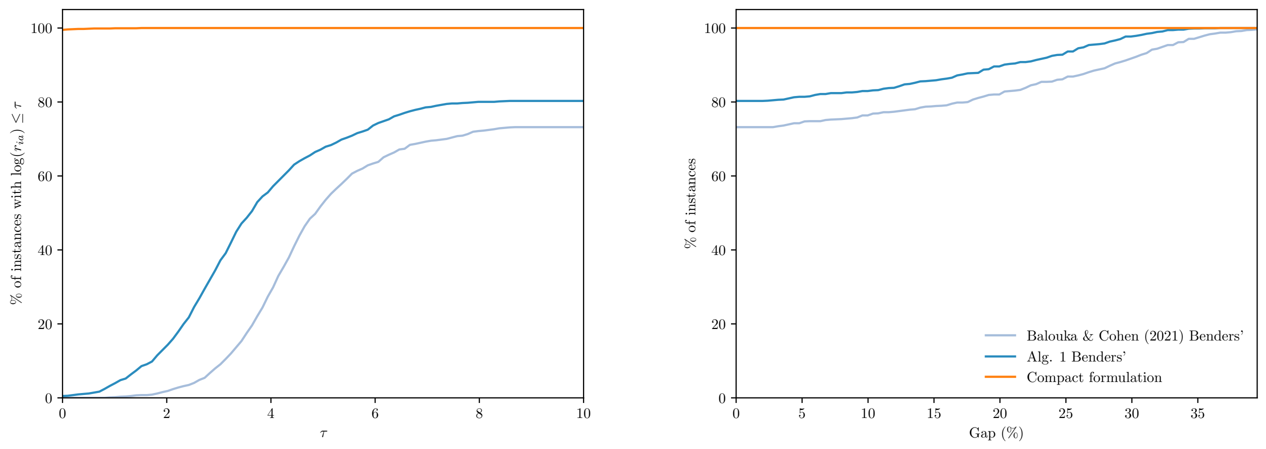
Firstly, the results in Table 6.3 show that for both methods, the instances tend to increase in difficulty as Γ increases. We can also see that for the nominal problems, i.e. when $\Gamma = 0$, the Benders' approach has a slight edge on the compact formulation. This is what we would expect given the relationship between the Benders' master problem formulation and the compact formulation. However, for all the non-zero values of Γ across both instance sets, the compact formulation performs significantly better than the Benders' approach, solving a greater proportion of instances with dramatically reduced computation times.

		Benders'					Compact formulation		
	Γ	*%sol.*	*gap*	*its.*	*it. time*	*sol. time*	*%sol.*	*gap*	*sol. time*
$j10$	0	100.0	0.00	1	0.96	1.1	100.0	0.00	1.8
	3	81.7	3.13	67	3.76	1551.4	100.0	0.00	5.0
	5	79.7	4.20	76	3.89	1688.4	100.0	0.00	4.6
	7	79.5	4.52	78	3.98	1701.1	100.0	0.00	6.5
		85.2	2.96	56	3.15	1235.5	100.0	0.00	4.47
$j20$	0	89.9	0.00	1	171.08	885.5	89.5	1.89	925.1
	5	50.9	10.21	220	133.32	4169.3	88.6	1.87	1041.5
	10	50.4	10.99	225	138.49	4244.6	87.9	2.58	1118.5
	15	49.5	11.21	216	146.26	4300.3	86.3	2.91	1260.6
		60.2	8.10	165	147.28	3399.9	88.1	2.31	1086.4

Table 6.3: Comparison of the Benders' decomposition approach and the compact reformulation across the $j10$ and $j20$ instance sets and for different values of Γ.

Figures 6.4 and 6.5 show performance profiles and optimality gap plots for the compact formulation and both implementations of the Benders' decomposition solution approach (Algorithm 6.1 and Balouka and Cohen (2021)), across the $j10$ and $j20$ sets respectively. Note that only the results for the instances with non-zero values for Γ were used to generate these plots, resulting in a total of 1608 $j10$ instances and 1664 $j20$ instances.

Performance profiles (Dolan and Moré (2002)) provide a visual comparison of competing solution approaches using their *performance ratios*. The performance



(a) Performance profiles showing percentage of instances with a performance ratio of within τ. Plotted on a log scale.

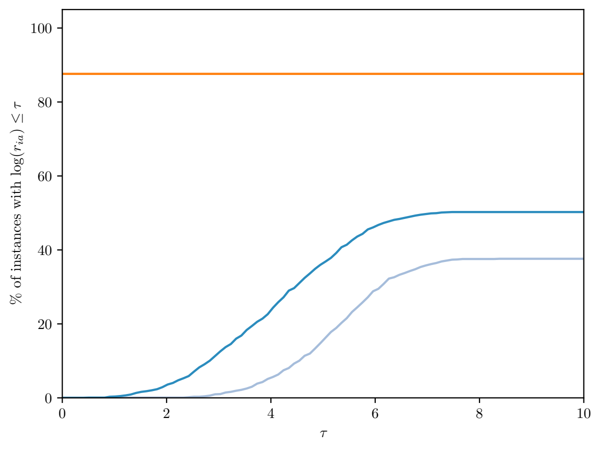
(b) Percentage of instances solved to within given optimality gap within the two-hour time limit.

Figure 6.4: Comparison of the Benders' approaches and compact formulation over instances in the $j10$ set.

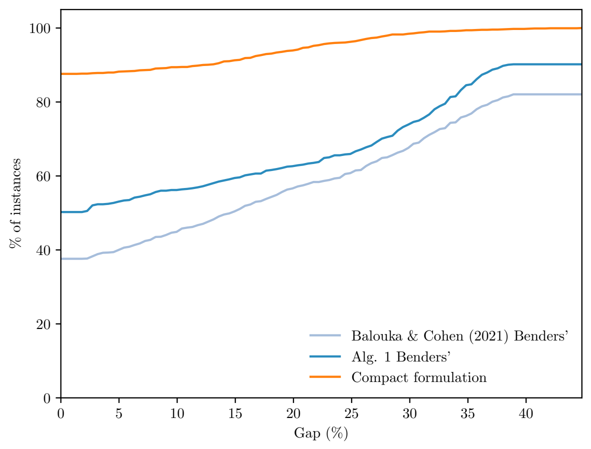
ratio of algorithm $a \in \mathcal{A}$ for instance $i \in \mathcal{I}$ is defined to be

$$r_{ia} = \frac{t_{ia}}{\min_{a \in \mathcal{A}} t_{ia}},$$

where t_{ia} is the time required to solve instance i using algorithm a. If method a is unable to solve an instance i within the 2-hour time limit, then $r_{ia} = R$, where $R \geq \max_{i,m} r_{im}$. The performance profiles in Figures 6.4a and 6.5a show the percentage of instances that have a performance ratio within a factor of τ of the best algorithm. These are plotted using a log scale for clarity. The y-intercepts of each method give the percentage of instances for which that method found the optimal solution in the fastest time, whereas the right-most value gives the overall percentage of instances that were solved to optimality within the time limit by



(a) Performance profile showing percentage of instances with a performance ratio of within τ. Plotted on a log scale.



(b) Percentage of instances solved to within given optimality gap within the two-hour time limit.

Figure 6.5: Comparison of the Benders' approaches and compact formulation over instances in the $j20$ set.

that method.

These performance profiles show that whenever an instance was able to be solved to optimality by either solution approach, the compact formulation was always the faster method. If we specifically compare the solution times of the compact formulation with the stronger of the two Benders' implementations, across the instances in $j10$ for which both methods found the optimal solution, the average solution time of the compact formulation (1.4s) was almost 200 times faster than for the Benders' approach (283.5s). When both methods found the optimal solution for instances in the $j20$ set, the average solution time of the compact formulation (13.7s) was almost 100 times faster than for the Benders' approach (1304.5s).

Figures 6.4b and 6.5b show the percentage of instances solved by each method to within a given optimality gap within the 2-hour time limit. These plots serve as a continuation of the performance profiles beyond just the instances that were solved to optimality. Summarising the data from these plots across both the $j10$ and $j20$ instance sets, the Balouka and Cohen (2021) Benders' implementation solves 55.1% of instances to optimality and finds feasible solutions for a further 35.6% of instances. Our Benders' implementation solves 65.0% of instances to optimality and finds feasible solutions for a further 30.0% of instances. The compact formulation on the other hand finds a feasible solution to every instance, solving 93.1% these to optimality.

The results presented here show the clear improvements to the Benders' algorithm afforded by amending the master problem to use the formulation (6.21)-(6.27). More significantly however, these results demonstrate the complete dominance of the compact formulation over both Benders' methods.

6.6 Conclusions

The work presented in this chapter extends the compact mixed-integer programming formulation introduced in the previous chapter (Bold and Goerigk, 2021), for application to the two-stage adjustable robust MRCPSP with uncertain activity durations. The computational performance of this formulation has been examined over a total of 3270 uncertain MRCPSP instances of varying size and difficulty, and compared against an improved version of the current state-of-the-art for solving this problem, based on a Benders' decomposition approach. The improved Benders' approach is the result of replacing the original master problem with a new formulation based on a simplified version of the compact formulation we present for the full problem.

Results presented in Section 6.5 show that the compact formulation completely dominates the enhanced Benders' approach, solving over 43% more instances to optimality, and doing so with dramatically reduced computation times. In addition to these strong computational improvements, the proposed compact formulation has the significant added benefit of being simpler to implement than the iterative Benders' approach.

Despite these strong results, the instances used in these experiments contain only up to 20 activities. To enable the solving of larger scale instances, the development of heuristic solution approaches should be a primary focus of future research on this problem.

Chapter 7

Recoverable Robust Single Machine Scheduling with Polyhedral Uncertainty

7.1 Introduction

We consider a scheduling problem where n jobs must be scheduled on a single machine without preemption, such that the total flow time, i.e. the sum of completion times, is minimised. This problem is denoted as $1||\sum C_i$ under the $\alpha|\beta|\gamma$ scheduling problem notation introduced by Graham et al. (1979). In practice, job processing times are often subject to uncertainty, and when this is the case it is important to find robust solutions that account for this uncertainty. In this paper, we propose a recoverable robust approach (Liebchen et al., 2009) to this uncertain single machine scheduling problem. In this recoverable robust setting, we determine a full solution in a first-stage, before an adversarial player chooses a worst-case scenario of processing times from an uncertainty set, and then in response to this, we allow the first-stage solution to be adjusted in a limited way.

The deterministic single machine scheduling problem (SMSP) is one of the simplest and most studied scheduling problems, and can be solved easily in $O(n \log n)$ time by ordering the jobs according to non-decreasing processing times, i.e. by using the shortest processing time (SPT) rule. However, despite the simplicity of the nominal problem, the robust problem has been shown to be NP-hard for even the most basic uncertainty sets (Daniels and Kouvelis, 1995).

In fact, the majority of research to date regarding robust single machine scheduling has been concerned with the presentation of complexity results for a number of different SMSPs. First discussed by Daniels and Kouvelis (1995), Kouvelis and Yu (1997) and Yang and Yu (2002), these papers study the problem with the total flow time objective, and show that it is NP-hard even in the case of two discrete scenarios, for min-max, regret and relative regret robustness. Robust single machine scheduling for discrete uncertain scenarios has been examined extensively. Aloulou and Della Croce (2008) present algorithmic and complexity results for a number of different SMSPs under min-max robustness. Aissi et al. (2011) show that the problem of minimising the number of late jobs in the worst-case scenario, where processing times are known but due dates are uncertain is NP-hard. Zhao et al. (2010) consider the objective of minimising the weighted sum of completion times in the worst-case scenario, and propose a cutting-plane algorithm to solve the problem. Mastrolilli et al. (2013) study this same problem and show that no polynomial-time approximation scheme exists for the unweighted version. Kasperski and Zieliński (2016b) apply the ordered weighted averaging (OWA) criterion, of which classical robustness is a special case, to a number of different SMSPs under discrete uncertainty. The consideration of SMSPs under novel optimality criteria has been continued most recently by Kasperski and Zieliński (2019), where a number of complexity results are presented for the SMSP with the value at risk (VaR) and conditional value at risk (CVaR) criteria.

Robust single machine scheduling in the context of interval uncertainty has also received considerable attention. Daniels and Kouvelis (1995) address interval uncertainty, and describe some dominance relations between the jobs in an optimal schedule based on their processing time intervals. Kasperski (2005) considers an SMSP with precedence constraints, where the regret of the maximum lateness of a job is minimised. A polynomial-time algorithm is presented. Lebedev and Averbakh (2006) show that the SMSP with the total flow time objective is NP-hard in the case of regret robustness. Montemanni (2007) presents a mixed-integer program (MIP) for this same problem, and use it to solve instances involving up to 45 jobs. Kasperski and Zieliński (2008) also consider this problem, and show that it is 2-approximable when the corresponding deterministic problem is polynomially solvable. Lu et al. (2012) present an SMSP with uncertain job processing and setup times, show this problem is NP-hard, and design a simulated annealing-based algorithm to solve larger instances. Chang et al. (2017) apply distributional robustness to an SMSP, and make use of information about the mean and covariance of the job processing times to minimise the worst-case CVaR. Most recently, Fridman et al. (2020) consider an SMSP with uncertain job processing times and develop polynomial algorithms for solving the min-max regret problem under certain classes of cost functions. For a survey of robust single-machine scheduling in the context of both discrete and interval uncertainty, see Kasperski and Zielinski (2014).

A criticism of classical robustness is that the solutions it provides are overly conservative and hedge against extreme worst-case scenarios that are very unlikely to occur in practice. To reduce the level of conservatism, a restriction to interval uncertainty was introduced by Bertsimas and Sim (2004), known as budgeted uncertainty, in which the number of jobs that can simultaneously achieve their worst-case processing times is restricted. Budgeted uncertainty is a special case of the general compact polyhedral uncertainty that is considered in this paper. Robust single machine scheduling under budgeted uncertainty was first considered by Lu et al. (2014), who present both an MIP and a heuristic to solve the problem. Following this, Tadayon and Smith (2015) study different versions of the min-max robust SMSP under three different uncertainty sets, including a budgeted uncertainty set. Recently, Bougeret et al. (2019) present complexity results and approximation algorithms for a number of different min-max robust scheduling problems under budgeted uncertainty.

To the best of our knowledge, the work presented in this chapter constitutes the first time a single-machine scheduling problem has been solved in a recoverable robust setting. However, recoverable robustness has had recent application to a number of closely related matching, assignment and scheduling problems. Fischer et al. (2020) consider a recoverable robust assignment problem, in which two perfect matchings of minimum costs must be chosen, subject to these matchings having at least k edges in common. If the cost of the second matching is evaluated in the worst-case scenario, we arrive in the setting of recoverable robustness with interval uncertainty. Hardness results are presented, and a polynomial-time algorithm is developed for the restricted case in which one cost function is Monge. The work presented in Chapter 8 of this thesis (published as Bold and Goerigk (2022a)) also considers recoverable robust scheduling problems under interval uncertainty, deriving a 2-approximation algorithm for that setting.

Regarding project scheduling, Bendotti et al. (2019) introduce the so-called anchor-robust project scheduling problem in which a baseline schedule is designed under the problem uncertainty, such that the largest possible subset of jobs have their starting times unchanged following the realisation of the activity processing times. This problem is shown to be NP-hard even for budgeted uncertainty. In a series of papers Bruni et al. (2017, 2018); Bold and Goerigk (2021), a two-stage resource-constrained project scheduling problem with budgeted uncertainty is introduced and solved.

The contributions of this chapter are as follows. In Section 7.2 we formally define the recoverable robust scheduling problem that we consider in this chapter. In Section 7.3 we present a general result that enables the construction of compact formulations for a wide range of recoverable robust problems, and apply this in the context of the scheduling problem at hand. We then analyse the stages of the recoverable robust scheduling problem in detail and show that the incremental problem can be solved using a simple linear programming formulation in Section 7.4. To this end, we prove a general result for max-weight matching problems, arguing that odd-cycle constraints are not required in problems with weights of a specific form. This formulation of the incremental problem then leads to an alternative matching-based compact formulation. Additionally, we transfer the matching result to an assignment-based formulation for the incremental problem, which results in a third compact model. In Section 7.6, computational experiments are presented, showing the benefits of a recourse action, the effects of the uncertainty on the model, and the strength of the assignment-based formulation. Finally, some concluding remarks and potential directions for future research are given in Section 7.7.

7.2 Problem definition

We consider a single machine scheduling problem with the objective of minimising the sum of completion times. Given a set of jobs $\mathcal{N} = \{1, \ldots, n\}$ with processing times $\boldsymbol{p} = (p_1, \ldots, p_n)$, we aim to find a schedule, i.e. an ordering of the jobs $i \in \mathcal{N}$, that minimises the sum of completion times. This nominal problem is denoted by $1||\sum C_i$ under the $\alpha|\beta|\gamma$ scheduling problem notation introduced by Graham et al. (1979). Recall that this problem is easy to solve; the shortest processing time (SPT) rule of sorting jobs by non-decreasing processing times results in an optimal schedule. This problem can be modelled as the following assignment problem with non-general costs:

$$\min \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} p_i(n+1-j)x_{ij} \tag{7.1}$$

$$\text{s.t.} \sum_{i \in \mathcal{N}} x_{ij} = 1 \qquad \forall j \in \mathcal{N} \tag{7.2}$$

$$\sum_{j \in \mathcal{N}} x_{ij} = 1 \qquad \forall i \in \mathcal{N} \tag{7.3}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,\, j \in \mathcal{N}, \tag{7.4}$$

where $x_{ij} = 1$ if job i is scheduled in position j, and $x_{ij} = 0$ otherwise.

We assume the job processing times p_i, $i \in \mathcal{N}$ are uncertain, but are known to lie within a given uncertainty set $\mathcal{U}$. In this paper, we consider a general polyhedral uncertainty set given by

$$\mathcal{U} = \left\{ \boldsymbol{p} \in \mathbb{R}^n_+ : \boldsymbol{A}\boldsymbol{p} \leq \boldsymbol{b} \right\},$$

where $\boldsymbol{A} \in \mathbb{R}^{M \times n}$ and $\boldsymbol{b} \in \mathbb{R}^M$, consisting of M linear constraints $a_{m1}p_1 + a_{m2}p_2 + \cdots + a_{mn}p_n \leq b_m$ for $m \in \mathcal{M} = \{1, \ldots, M\}$ on the set of possible processing times $\boldsymbol{p}$. Throughout this paper, we assume $\mathcal{U}$ to be compact. It is also possible to include auxiliary variables in the definition of $\mathcal{U}$; for ease of presentation, such variables have been omitted.

We consider this uncertain single machine scheduling problem in the context of a two-stage decision process, where, having decided on a first-stage schedule $\boldsymbol{x}$ under the problem uncertainty, the decision-maker is given the opportunity to react to the realisation of the uncertain data by choosing up to Δ distinct pairs of jobs and swapping their positions, to obtain a second-stage schedule $\boldsymbol{y}$.

This recoverable robust problem can be written as follows:

$$\min_{\boldsymbol{x} \in \mathcal{X}} \max_{\boldsymbol{p} \in \mathcal{U}} \min_{\boldsymbol{y} \in \mathcal{X}(\boldsymbol{x})} \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} p_i (n + 1 - j) y_{ij}, \tag{RRS}$$

where $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^{n \times n} : (7.2), (7.3)\}$ is the set of feasible schedules, and $\mathcal{X}(\boldsymbol{x}) \subseteq \mathcal{X}$ is the set of feasible second-stage assignments given $\boldsymbol{x}$. That is,

$$\mathcal{X}(\boldsymbol{x}) = \{\boldsymbol{y} \in \mathcal{X} : d(\boldsymbol{x}, \boldsymbol{y}) \leq \Delta\},$$

where $d(\boldsymbol{x}, \boldsymbol{y})$ is some measure of the distance between the first and second-stage schedules.

In this paper, we restrict our attention to the case where the recourse action consists of disjoint pairwise swaps to the first-stage positions of the jobs. This choice of recourse can be motivated with an example. Consider a nuclear storage silo full of ageing containers of untreated nuclear waste that each must undergo a reprocessing procedure to treat the waste and place it into new, safe long-term storage containers. This process is scheduled in advance of its execution and its management is allocated among a number of different store managers. For many of the old waste containers, it is unclear which grade of waste they contain and therefore how extensive their reprocessing procedure will be. After the creation of the preliminary schedule, the nuclear material in each container is examined in detail and the reprocessing time estimates are improved. Having obtained these improved estimates, the reprocessing schedule can be updated. If the tasks that a store manager is responsible for change in the updated schedule, they must meet with the manager that was previously responsible for those tasks in order to be briefed about their technical requirements. To simplify the arrangements of such meetings so that store managers only have to swap briefings with a single other store manager, we restrict schedule updates to disjoint pairwise swaps of waste processing jobs.

In addition to this motivation, the restriction to disjoint pairwise swaps improves the tractability of the problem and leads to the results that we present and analyse in this paper.

Hence, in this case we define $d(\boldsymbol{x}, \boldsymbol{y})$ to be the minimum number of pairwise distinct swaps required to transform $\boldsymbol{x}$ into $\boldsymbol{y}$, if this number exists; otherwise, we set it to ∞. Observe that the value $d(\boldsymbol{x}, \boldsymbol{y})$ can be calculated using the following approach. Let $\mathcal{E}_{\boldsymbol{x}}$ be the edges chosen by $\boldsymbol{x}$ in the corresponding bipartite graph, oriented towards the right, and let $\mathcal{E}_{\boldsymbol{y}}$ be the edges chosen by $\boldsymbol{y}$, oriented towards the left, i.e. $(j,i) \in \mathcal{E}_{\boldsymbol{y}}$ corresponds to assigning job i to position j. If and only if the edges $\mathcal{E}_{\boldsymbol{x}} \cup \mathcal{E}_{\boldsymbol{y}}$ decompose into 2-cycles and 4-cycles, we have $d(\boldsymbol{x},\boldsymbol{y}) < \infty$, in which case $d(\boldsymbol{x},\boldsymbol{y})$ is equal to the number of 4-cycles. This is because a 2-cycle corresponds to a job with an unchanged position, whilst 4-cycles represent a swap of positions of two jobs. An example is given in Figure 7.1.

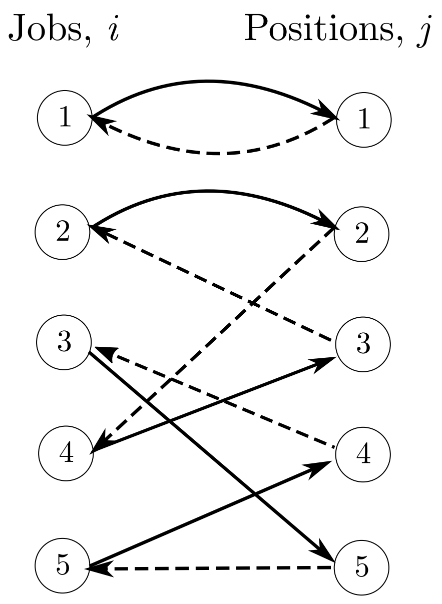


Figure 7.1: An example first and second-stage solution. The first-stage assignment is given by the solid arcs oriented towards the right, and corresponds to the schedule (1,2,4,5,3). The second-stage assignment is given by the dashed arcs oriented towards the left, and corresponds to the schedule (1,4,2,3,5). There are two 4-cycles corresponding to the switching of positions of jobs 2 and 4, and 3 and 5. Hence $d(\boldsymbol{x},\boldsymbol{y}) = 2$.

We define the adversarial and incremental problems of (RRS) as follows. Given both a first-stage solution $\boldsymbol{x} \in \mathcal{X}$ and a scenario $\boldsymbol{p} \in \mathcal{U}$, the incremental problem consists of finding the best possible second-stage solution $\boldsymbol{y} \in \mathcal{X}(\boldsymbol{x})$. That is,

$$\text{Inc}(\boldsymbol{x}, \boldsymbol{p}) = \min_{\boldsymbol{y} \in \mathcal{X}(\boldsymbol{x})} \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} p_i(n+1-j)y_{ij}.$$

The adversarial problem is to find a worst-case scenario $\boldsymbol{p} \in \mathcal{U}$ for a given first-stage schedule $\boldsymbol{x} \in \mathcal{X}$. That is,

$$\text{Adv}(\boldsymbol{x}) = \max_{\boldsymbol{p} \in \mathcal{U}} \min_{\boldsymbol{y} \in \mathcal{X}(\boldsymbol{x})} \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} p_i(n+1-j)y_{ij} = \max_{\boldsymbol{p} \in \mathcal{U}} \text{Inc}(\boldsymbol{x}, \boldsymbol{p}).$$

Observe that for the case of general polyhedral uncertainty that we consider here, (RRS) is NP-hard. To see this, suppose that $\Delta = 0$, i.e. there is no recovery option and the second-stage variables are fixed to the corresponding first-stage values. Then the problem reduces to a standard robust single machine scheduling problem of the form

$$\min_{\boldsymbol{x} \in \mathcal{X}} \max_{\boldsymbol{p} \in \mathcal{U}} \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} p_i(n+1-j)x_{ij}.$$

Note that a general polyhedral uncertainty set can be used to construct a problem involving only two discrete scenarios. This can be done by simply defining the uncertainty set to be the linear combination of the two discrete points, since the worst-case scenario must lie at a vertex of the polyhedron, i.e. at one of the two discrete scenarios. Since the robust scheduling problem with two scenarios is already NP-hard (see Kouvelis and Yu (1997)), this hardness result also extends to our setting.

Finally, note that in problem (RRS) we aim to minimise the worst-case costs of the resulting recovery solutions. If the first-stage costs are also relevant, all the results presented in this paper can be adjusted trivially.

7.3 A general model for recoverable robustness

In this section we present a general model for recoverable robust optimisation problems, and apply this method to the uncertain single machine scheduling problem (RRS). Our approach is to determine a first-stage solution $\boldsymbol{x} \in \mathcal{X}$ as well as a finite set of candidate recovery solutions $\boldsymbol{y}^1, \dots, \boldsymbol{y}^K \in \mathcal{X}(\boldsymbol{x})$.

The following result shows that $K = n^2 + 1$ is sufficient to guarantee that this approach provides an exact solution to the problem.

Theorem 7.1. *Let a recoverable robust problem of the form*

$$\min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{c}\in\mathcal{U}} \min_{\boldsymbol{y}\in\mathcal{X}(\boldsymbol{x})} f(\boldsymbol{y}, \boldsymbol{c})$$

be given, where $\mathcal{X}, \mathcal{X}(\boldsymbol{x}) \subseteq \{0,1\}^n$, $\mathcal{U}$ is a compact convex set, f is linear in $\boldsymbol{y}$, and concave in $\boldsymbol{c}$. Then this problem is equivalent to

$$\min_{\substack{\boldsymbol{x}\in\mathcal{X},\\ \boldsymbol{y}^{(1)},\dots,\boldsymbol{y}^{(n+1)}\in\mathcal{X}(\boldsymbol{x})}} \max_{\boldsymbol{c}\in\mathcal{U}} \min_{i=1,\dots,n+1} f(\boldsymbol{y}^{(i)}, \boldsymbol{c}).$$

Proof. The idea of the proof is similar to models developed for K-adaptability (see Hanasusanto et al. (2015, Theorem 1) and Buchheim and Kurtz (2017, Corollary 1)). Recall both Carathéodory's theorem and the minimax theorem. Carathéodory's theorem states that any point $x \in \mathbb{R}^n$ lying in $conv(X)$ can be written as a convex combination of $n+1$ points from X. The minimax theorem states that if X and Y

are two compact, convex sets, and $f : X \times Y \to \mathbb{R}$ is a continuous compact-concave function (i.e. $f(\cdot, y)$ is concave for fixed values of y and $f(x, \cdot)$ is convex for fixed values of x), then

$$\max_x \min_y f(x,y) = \min_y \max_x f(x,y).$$

We make use of both of these results in the following:

$$\begin{aligned}
&\min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{c}\in\mathcal{U}} \min_{\boldsymbol{y}\in\mathcal{X}(\boldsymbol{x})} f(\boldsymbol{y},\boldsymbol{c}) \\
&= \min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{c}\in\mathcal{U}} \min_{\boldsymbol{y}\in conv(\mathcal{X}(\boldsymbol{x}))} f(\boldsymbol{y},\boldsymbol{c}) \\
&= \min_{\boldsymbol{x}\in\mathcal{X}} \min_{\boldsymbol{y}\in conv(\mathcal{X}(\boldsymbol{x}))} \max_{\boldsymbol{c}\in\mathcal{U}} f(\boldsymbol{y},\boldsymbol{c}) && \text{(by the minimax theorem)} \\
&= \min_{\boldsymbol{x}\in\mathcal{X}} \min_{\boldsymbol{y}^{(1)},\dots,\boldsymbol{y}^{(n+1)}\in\mathcal{X}(\boldsymbol{x})} \min_{\substack{\lambda^1,\dots,\lambda^{n+1}\geq 0\\ \sum_{i=1}^{n+1}\lambda^i=1}} \max_{\boldsymbol{c}\in\mathcal{U}} f(\sum_{i=1}^{n+1}\lambda^i\boldsymbol{y}^{(i)},\boldsymbol{c}) && \text{(by Carathéodory's theorem)} \\
&= \min_{\boldsymbol{x}\in\mathcal{X}} \min_{\boldsymbol{y}^{(1)},\dots,\boldsymbol{y}^{(n+1)}\in\mathcal{X}(\boldsymbol{x})} \max_{\boldsymbol{c}\in\mathcal{U}} \min_{\substack{\lambda^1,\dots,\lambda^{n+1}\geq 0\\ \sum_{i=1}^{n+1}\lambda^i=1}} f(\sum_{i=1}^{n+1}\lambda^i\boldsymbol{y}^{(i)},\boldsymbol{c}) && \text{(by the minimax theorem)} \\
&= \min_{\boldsymbol{x}\in\mathcal{X}} \min_{\boldsymbol{y}^{(1)},\dots,\boldsymbol{y}^{(n+1)}\in\mathcal{X}(\boldsymbol{x})} \max_{\boldsymbol{c}\in\mathcal{U}} \min_{\substack{\lambda^1,\dots,\lambda^{n+1}\geq 0\\ \sum_{i=1}^{n+1}\lambda^i=1}} \sum_{i=1}^{n+1}\lambda^i f(\boldsymbol{y}^{(i)},\boldsymbol{c}) \\
&= \min_{\substack{\boldsymbol{x}\in\mathcal{X},\\ \boldsymbol{y}^{(1)},\dots,\boldsymbol{y}^{(n+1)}\in\mathcal{X}(\boldsymbol{x})}} \max_{\boldsymbol{c}\in\mathcal{U}} \min_{i=1,\dots,n+1} f(\boldsymbol{y}^{(i)},\boldsymbol{c}).
\end{aligned}$$

□

This approach can be used to derive a compact formulation to the uncertain single machine scheduling problem (RRS). To this end, we first consider the inner selection problem, given a first-stage solution $\boldsymbol{x}$ and set of recovery solutions $\boldsymbol{y}^1, \dots, \boldsymbol{y}^K$, and a scenario $\boldsymbol{p}$. This is given by

$$\min \sum_{k\in\mathcal{K}} \left(\sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}} p_i(n+1-j) y_{ij}^k \right) \lambda_k$$

$$\begin{aligned}
\text{s.t. } & \sum_{k\in\mathcal{K}} \lambda_k = 1 \\
& \lambda_k \geq 0 && \forall k \in \mathcal{K},
\end{aligned}$$

where $\mathcal{K} = \{1, \dots, K\}$. The problem of finding the worst-case scenario $\boldsymbol{p} \in \mathcal{U}$ for the choice of first-stage solution $\boldsymbol{x}$ and recovery solutions $\boldsymbol{y}^1, \dots, \boldsymbol{y}^K$ is therefore:

$$\begin{aligned}
\max\ & t \\
\text{s.t. } & t \leq \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} p_i(n+1-j)y_{ij}^k && \forall k \in \mathcal{K} \\
& \sum_{i\in\mathcal{N}} a_{mi}p_i \leq b_m && \forall m \in \mathcal{M} \\
& p_i \geq 0 && \forall i \in \mathcal{N}.
\end{aligned}$$

Dualising this problem then gives the following formulation for (RRS):

$$\begin{aligned}
\min\ & \sum_{m\in\mathcal{M}} b_m q_m && && (7.5) \\
\text{s.t. } & \sum_{k\in\mathcal{K}} \mu_k = 1 && && (7.6) \\
& \sum_{m\in\mathcal{M}} a_{mi}q_m \geq \sum_{k\in\mathcal{K}} \left(\sum_{j\in\mathcal{N}} (n+1-j)y_{ij}^k \right) \mu_k && \forall i \in \mathcal{N} && (7.7) \\
& d(\boldsymbol{x}, \boldsymbol{y}^k) \leq \Delta && \forall k \in \mathcal{K} && (7.8) \\
& \boldsymbol{x} \in \mathcal{X} && && (7.9) \\
& \boldsymbol{y}^k \in \mathcal{X} && \forall k \in \mathcal{K} && (7.10) \\
& \mu_k \geq 0 && \forall k \in \mathcal{K} && (7.11) \\
& q_m \geq 0 && \forall m \in \mathcal{M}, && (7.12)
\end{aligned}$$

where $d(\boldsymbol{x}, \boldsymbol{y}^k)$ is some measure of distance between the first-stage solution and the k-th recovery solution. Note that this model is not restricted to any particular choice of distance measure $d(\boldsymbol{x}, \boldsymbol{y}^k)$.

However, if we opt to calculate the distance between two schedules as the minimum number of disjoint pairwise swaps required to transform one schedule into the other, this can be modelled as follows. Let $z^k_{ii'}$ denote the whether or not jobs i and i' have swapped positions in recovery solution $\boldsymbol{y}^k$, relative to the first-stage schedule $\boldsymbol{x}$. In this case, we have that

$$y^k_{ij} = \sum_{i' \in \mathcal{N}} z^k_{ii'} x_{i'j}.$$

Hence, $\boldsymbol{y}^k$ can be removed from the model, and replaced by $\boldsymbol{z}^k$ with the inclusion of the following constraints:

$$\begin{aligned}
&\sum_{i' \in \mathcal{N}} z^k_{ii'} = 1 && \forall i \in \mathcal{N},\, k \in \mathcal{K}\\
&\sum_{i \in \mathcal{N}} z^k_{ii'} = 1 && \forall i' \in \mathcal{N},\, k \in \mathcal{K}\\
&z^k_{ii'} = z^k_{i'i} && \forall i, i' \in \mathcal{N},\, k \in \mathcal{K}\\
&\sum_{i \in \mathcal{N}} z^k_{ii} \geq n - 2\Delta && \forall k \in \mathcal{K}.
\end{aligned}$$

To arrive at a mixed-integer linear program, the products $z^k_{ii'} \cdot x_{i'j} \cdot \mu_k$ need to be linearised using standard techniques. The full linearised formulation contains $O(n^3K)$ constraints and variables and is shown in Appendix C.2.1.

7.4 Complexity of subproblems and compact formulations

In this section, we examine the incremental and adversarial problems of (RRS) in more detail and subsequently derive two additional compact formulations.

7.4.1 Matching-based formulation

We first consider a matching-based formulation for the incremental problem. For the ease of presentation, we assume for now that $x_{ii} = 1$ for all $i \in \mathcal{N}$, i.e. the first-stage solution is a horizontal matching. Note a change in notation for this section where now the indices i and j are both used to refer to jobs, and ℓ denotes a position in the schedule. Supposing that the positions of jobs i and j are switched in the recovery schedule, the reduction in cost of making this switch is given by

$$p_i(n+1-i) + p_j(n+1-j) - p_i(n+1-j) - p_j(n+1-i) = (p_i - p_j)(j-i).$$

Letting z_{ij} indicate whether or not jobs i and j swap positions in the schedule, the incremental problem can be formulated as:

$$\min \sum_{i\in\mathcal{N}} p_i(n+1-i) - \sum_{e=\{i,j\}\in\mathcal{E}} (p_i - p_j)(j-i)z_e \tag{7.13}$$

$$\text{s.t.} \sum_{e\in\delta(i)} z_e \leq 1 \qquad \forall i \in \mathcal{N} \tag{7.14}$$

$$\sum_{e\in\mathcal{E}} z_e \leq \Delta \tag{7.15}$$

$$z_e \in \{0,1\} \qquad \forall e \in \mathcal{E}, \tag{7.16}$$

where $\mathcal{E} = \{\{i,j\} : i,j \in \mathcal{N},\ i \neq j\}$ is the set of unique swaps, and $\delta(i)$ is the set of edges incident to vertex i. This is a cardinality-constrained matching problem on a complete graph with one node for each job $i \in \mathcal{N}$.

We examine this matching-based formulation in further detail. First, consider the maximum weight matching problem on a general graph $G = (\mathcal{V}, \mathcal{E})$. This problem can be formulated as the following linear program:

$$\max \sum_{e\in\mathcal{E}} w_e x_e \tag{7.17}$$

$$\text{s.t.} \sum_{e\in\delta(i)} x_e \leq 1 \qquad \forall i \in \mathcal{V} \tag{7.18}$$

$$\sum_{e\in\mathcal{E}(\mathcal{W})} x_e \leq \frac{|\mathcal{W}|-1}{2} \qquad \forall \mathcal{W} \subseteq \mathcal{V},\ |\mathcal{W}| \text{ odd} \tag{7.19}$$

$$x_e \geq 0 \qquad \forall e \in \mathcal{E}, \tag{7.20}$$

where $\mathcal{E}(\mathcal{W})$ is the set of edges in the subgraph induced on $\mathcal{W}$. Edmonds (1965) showed that constraints (7.19), known as odd-cycle constraints or blossom constraints, are required to fully characterise the matching polytope.

In the following theorem, we show that for a matching problem with the same cost structure as (7.13), odd-cycle constraints are not required.

Theorem 7.2. *For any $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}_+^{|\mathcal{V}|}$, the problem*

$$\max \sum_{e=\{i,j\}\in\mathcal{E}} (a_i - a_j)(b_i - b_j) x_e \tag{7.21}$$

$$s.t. \sum_{e\in\delta(i)} x_e \leq 1 \qquad \forall i \in \mathcal{V} \tag{7.22}$$

$$x_e \geq 0 \qquad \forall e \in \mathcal{E} \tag{7.23}$$

has an optimal solution with $x_e \in \{0, 1\}$ *for all* $e \in \mathcal{E}$*.*

Proof. Schrijver (2003, Theorem 30.2, page 522) states that each vertex of the matching polytope described by (7.22) and (7.23) is half-integer, i.e. $x_e \in \{0, \frac{1}{2}, 1\}$ for all $e \in \mathcal{E}$ in an optimal solution. Additionally, as observed by Balinski (1965), the vertices of the matching polytope can be partitioned into a matching $\mathcal{P}$, where $x_e = 1$ for each $e \in \mathcal{P}$, and a set of 1/2-fractional cycles of odd length, where $x_e = \frac{1}{2}$ for each e in the odd cycles. Hence, we can restrict our attention only to 1/2-fractional odd cycles, and show that there is an optimal solution where such cycles do not exist.

Suppose we are given an optimal solution containing a 1/2-fractional odd cycle, consisting of edges $\mathcal{C} = \{e_{i_1,i_2}, e_{i_2,i_3}, \ldots, e_{i_{q-1},i_q}, e_{i_q,i_1}\}$, with weights given by $w_{ij} = (a_i - a_j)(b_i - b_j)$. Without loss of generality, we assume an orientation in the cycle, where edges are directed as (i_j, i_{j+1}) for $j = 1, \ldots, q$, where $i_{q+1} = i_1$.

Note that if $w_e \leq 0$ for some edge e, it can be removed from $\mathcal{E}$, as such an edge will never be selected in an optimal matching. Hence, we may assume that $w_e > 0$ for all $e \in \mathcal{C}$. Since $w_{ij} = (a_i - a_j)(b_i - b_j) > 0$ for all $e_{ij} \in \mathcal{C}$, $(a_i - a_j)$ and $(b_i - b_j)$ must have the same sign. That is, either $a_i > a_j$ and $b_i > b_j$, in which case we refer to e_{ij} as a *decreasing edge*, or $a_i < a_j$ and $b_i < b_j$, in which case we refer to e_{ij} as an *increasing edge*.

We show that there is an optimal 1/2-fractional cycle that alternates between increasing and decreasing edges. Suppose that there are $p < q$ consecutive decreasing edges in $\mathcal{C}$, $e_{j_1,j_2}, e_{j_2,j_3}, \ldots, e_{j_{p-1},j_p}$, i.e. $a_{j_1} > a_{j_2} > \cdots > a_{j_p}$ and

$b_{j_1} > b_{j_2} > \cdots > b_{j_p}$. In this case

$$\begin{aligned}
w_{j_1,j_p} &= (a_{j_1} - a_{j_p})(b_{j_1} - b_{j_p}) \\
&= \Big((a_{j_1} - a_{j_2}) + (a_{j_2} - a_{j_3}) + \cdots + (a_{j_p-1} - a_{j_p})\Big) \\
&\qquad \cdot \Big((b_{j_1} - b_{j_2}) + (b_{j_2} - b_{j_3}) + \cdots + (b_{j_{p-1}} - b_{j_p})\Big) \\
&= w_{j_1,j_2} + w_{j_2,j_3} + \cdots + w_{j_{p-1},j_p} + (a_{j_1} - a_{j_2})\Big((b_{j_2} - b_{j_3}) + \cdots + (b_{j_{p-1}} - b_{j_p})\Big) \\
&\qquad + (a_{j_2} - a_{j_3})\Big((b_{j_1} - b_{j_2}) + \cdots + (b_{j_{p-1}} - b_{j_p})\Big) \\
&\qquad + \dots \\
&\qquad + (a_{j_{p-1}} - a_{j_p})\Big((b_{j_1} - b_{j_2}) + \cdots + (b_{j_{p-2}} - b_{j_{p-1}})\Big) \\
&> w_{j_1,j_2} + w_{j_2,j_3} + \cdots + w_{j_{p-1},j_p},
\end{aligned}$$

which means that replacing the p consecutive decreasing edges in $\mathcal{C}$ by the edge e_{j_1,j_p} would lead to an even better objective value (see Figure 7.2 for an illustration). The same argument can be used to show that there also cannot be p consecutive increasing edges in an optimal 1/2-fractional cycle.

We have therefore constructed an optimal 1/2-fractional cycle that strictly alternates between increasing and decreasing edges. Clearly, this is only possible if q is even. Since a 1/2-fractional even cycle can be written as a convex combination of two feasible matchings, this proves that there exists an optimal solution without any 1/2-fractional cycles. □

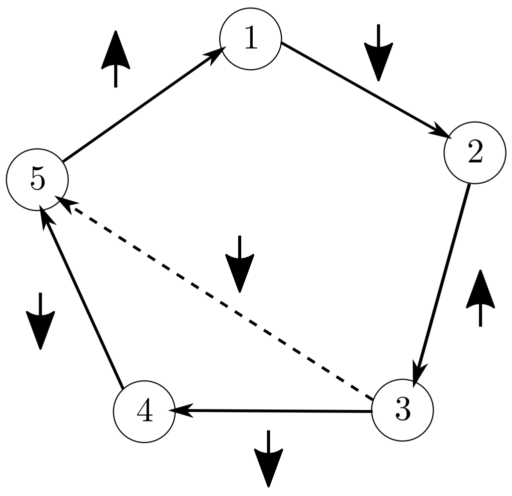


Figure 7.2: An example of a 1/2-fractional cycle involving $q = 5$ nodes. Up and down arrows indicate increasing and decreasing edges respectively. It is optimal to replace the two consecutive decreasing edges $(3, 4)$ and $(4, 5)$ with the dashed edge $(3, 5)$, i.e. $w_{35} > w_{34} + w_{45}$.

The following result, presented in Schrijver (2003) (Corollary 18.10a, page 331), states that the integrality of the vertices of the matching polytope is unaffected by the addition of a cardinality constraint.

Theorem 7.3. *Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected graph and let $k, l \in \mathbb{Z}_+$ with $k \leq l$. Then the convex hull of the incidence vectors of matchings $\mathcal{P}$ satisfying $k \leq |\mathcal{P}| \leq l$ is equal to the set of those vectors $\boldsymbol{x}$ in the matching polytope of G satisfying $k \leq \mathbf{1}^\top \boldsymbol{x} \leq l$.*

This result, in combination with Theorem 7.2, provides us with the following corollary:

Corollary 7.4. *For any $\boldsymbol{a}$, $\boldsymbol{b} \in \mathbb{R}_+^{|\mathcal{V}|}$, the problem*

$$\max \sum_{e=\{i,j\}\in\mathcal{E}} (a_i - a_j)(b_i - b_j)x_e \tag{7.24}$$

$$s.t. \sum_{e \in \delta(i)} x_e \leq 1 \qquad \forall i \in \mathcal{V} \tag{7.25}$$

$$\sum_{e \in \mathcal{E}} x_e \leq \Delta \tag{7.26}$$

$$x_e \geq 0 \qquad \forall e \in \mathcal{E} \tag{7.27}$$

has an optimal solution with $x_e \in \{0, 1\}$ *for all* $e \in \mathcal{E}$.

Hence, given a first-stage solution $\boldsymbol{x}$ and scenario $\boldsymbol{p}$, we can formulate the incremental problem as a linear program with polynomially many constraints. We use this result to derive a compact formulation for the full uncertain single machine scheduling problem (RRS).

We begin by formulating the incremental problem $\mathrm{Inc}(\boldsymbol{x}, \boldsymbol{p})$ according to Corollary 7.4. Note that we now consider a general first-stage assignment that is not necessarily horizontal, and therefore introduce terms $\sum_{\ell \in \mathcal{N}} \ell \cdot x_{i\ell}$ to track the position in which job i is scheduled in the first-stage schedule. We fix an arbitrary orientation of edges, using $\mathcal{E} = \{(i, j) \in \mathcal{N} \times \mathcal{N} : i < j\}$ in the following.

$$\begin{aligned}
\min_{\boldsymbol{z}} \ & \sum_{i \in \mathcal{N}} p_i \left(n + 1 - \sum_{\ell \in \mathcal{N}} \ell \cdot x_{i\ell} \right) - \sum_{(i,j) \in \mathcal{E}} (p_i - p_j) \left(\sum_{\ell \in \mathcal{N}} \ell \cdot x_{j\ell} - \sum_{\ell \in \mathcal{N}} \ell \cdot x_{i\ell} \right) z_{ij} \\
\text{s.t.} \ & \sum_{(i,j) \in \mathcal{E}} z_{ij} + \sum_{(j,i) \in \mathcal{E}} z_{ji} \leq 1 && \forall i \in \mathcal{N} \\
& \sum_{(i,j) \in \mathcal{E}} z_{ij} \leq \Delta \\
& z_{ij} \geq 0 && \forall (i, j) \in \mathcal{E}.
\end{aligned}$$

Taking the dual of this, we get the following formulation for the adversarial problem

Adv($\boldsymbol{x}$):

$$\begin{aligned}
\max_{\boldsymbol{p},\boldsymbol{\alpha},\gamma}\ & \sum_{i\in\mathcal{N}} p_i\left(n+1-\sum_{\ell\in\mathcal{N}}\ell\cdot x_{i\ell}\right)-\sum_{i\in\mathcal{N}}\alpha_i-\gamma\Delta \\
\text{s.t. }\ & \alpha_i+\alpha_j+\gamma\geq(p_i-p_j)\left(\sum_{\ell\in\mathcal{N}}\ell\cdot x_{j\ell}-\sum_{\ell\in\mathcal{N}}\ell\cdot x_{i\ell}\right) && \forall(i,j)\in\mathcal{E} \\
& \sum_{i\in\mathcal{N}} a_{mi}p_i\leq b_m && \forall m\in\mathcal{M} \\
& p_i\geq 0 && \forall i\in\mathcal{N} \\
& \alpha_i\geq 0 && \forall i\in\mathcal{N} \\
& \gamma\geq 0.
\end{aligned}$$

Since this is a linear program, we immediately obtain following result:

Corollary 7.5. *The adversarial problem can be solved in polynomial time.*

Finally, dualising the above adversarial formulation, we get the following compact formulation for problem (RRS):

$$\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{z},\boldsymbol{q}}\ & \sum_{m\in\mathcal{M}} b_m q_m && (7.28) \\
\text{s.t. }\ & \sum_{m\in\mathcal{M}} a_{mi}q_m+\sum_{(i,j)\in\mathcal{E}}\left(\sum_{\ell\in\mathcal{N}}\ell\cdot x_{j\ell}-\sum_{\ell\in\mathcal{N}}\ell\cdot x_{i\ell}\right)z_{ij} \\
& \qquad -\sum_{(j,i)\in\mathcal{E}}\left(\sum_{\ell\in\mathcal{N}}\ell\cdot x_{i\ell}-\sum_{\ell\in\mathcal{N}}\ell\cdot x_{j\ell}\right)z_{ji} \\
& \qquad\qquad \geq(n+1-\sum_{\ell\in\mathcal{N}}\ell\cdot x_{i\ell}) && \forall i\in\mathcal{N}\quad (7.29) \\
& \sum_{(i,j)\in\mathcal{E}} z_{ij}+\sum_{(j,i)\in\mathcal{E}} z_{ji}\leq 1 && \forall i\in\mathcal{N}\quad (7.30) \\
& \sum_{(i,j)\in\mathcal{E}} z_{ij}\leq\Delta && (7.31)
\end{aligned}$$

$$x \in \mathcal{X} \tag{7.32}$$

$$q_m \geq 0 \qquad \forall m \in \mathcal{M} \tag{7.33}$$

$$z_{ij} \geq 0 \qquad \forall (i,j) \in \mathcal{E}. \tag{7.34}$$

Upon linearising the quadratic $x_{i\ell} \cdot z_{ij}$ and $x_{j\ell} \cdot z_{ij}$ terms, this model becomes a mixed-integer linear program. The fully linearised model contains $O(n^3)$ constraints and variables and is presented in full in Appendix C.2.2.

7.4.2 Assignment-based formulation

We now consider an alternative formulation for the incremental problem. Again, for the purposes of examining the incremental problem, we initially consider the first-stage schedule to be a horizontal assignment, i.e. $x_{ii} = 1$ for all $i \in \mathcal{N}$. By letting variables y_{ij} represent a second-stage assignment (we now return to the convention where the index i is used to denote a job and the index j is used to denote a position in the schedule), we can formulate the incremental problem as follows:

$$\min \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} p_i(n+1-j) y_{ij} \tag{7.35}$$

$$\text{s.t.} \sum_{i \in \mathcal{N}} y_{ij} = 1 \qquad \forall j \in \mathcal{N} \tag{7.36}$$

$$\sum_{j \in \mathcal{N}} y_{ij} = 1 \qquad \forall i \in \mathcal{N} \tag{7.37}$$

$$y_{ij} = y_{ji} \qquad \forall i,j \in \mathcal{N} \tag{7.38}$$

$$\sum_{i \in \mathcal{N}} y_{ii} \geq n - 2\Delta \tag{7.39}$$

$$y_{ij} \in \{0,1\} \qquad\qquad \forall i,j \in \mathcal{N}. \tag{7.40}$$

Constraints (7.38) and (7.39) ensure that the second-stage assignment is a feasible recovery to the first-stage solution, that is, the second-stage assignment is constructed by swapping the first-stage positions of up to Δ disjoint pairs of jobs. Note that this is a level-constrained symmetric perfect matching problem, which can be solved in polynomial time (Thomas, 2015, Theorem 2.28).

We show that problem (7.35)-(7.40) can be solved as a linear program as a result of its non-general cost structure. As the proof is technical and based on a reduction to the corresponding maximum weight matching problem, it is omitted here and can be found in Appendix C.1.1.

Theorem 7.6. *For any $\boldsymbol{a}$, $\boldsymbol{b} \in \mathbb{R}^n_+$, the problem*

$$\min \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} a_i b_j y_{ij} \tag{7.41}$$

$$s.t.\ (7.36)-(7.39) \tag{7.42}$$

$$y_{ij} \geq 0 \tag{7.43}$$

has an optimal solution with $y_{ij} \in \{0,1\}$ for all $i,j \in \mathcal{N}$.

We now use this result to find an assignment-based formulation for (RRS). We first write the incremental problem in the form given by (7.41)-(7.43). Since we are now considering the case where $\boldsymbol{x}$ is not necessarily a horizontal matching, we rearrange the indices accordingly.

$$\min \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} p_i(n+1-\sum_{\ell\in\mathcal{N}} \ell \cdot x_{j\ell})y_{ij}$$

$$\begin{aligned}
\text{s.t. } & \sum_{i\in\mathcal{N}} y_{ij} = 1 && \forall j \in \mathcal{N}\\
& \sum_{j\in\mathcal{N}} y_{ij} = 1 && \forall i \in \mathcal{N}\\
& y_{ij} = y_{ji} && \forall i,j \in \mathcal{N}\\
& \sum_{i\in\mathcal{N}} y_{ii} \geq n - 2\Delta\\
& y_{ij} \geq 0 && \forall i,j \in \mathcal{N}.
\end{aligned}$$

Taking the dual of this, the adversarial problem can be formulated in the following way:

$$\begin{aligned}
\max_{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\tau,\boldsymbol{p}} \; & \sum_{i\in\mathcal{N}}(\alpha_i + \beta_i) + (n-2\Delta)\tau\\
\text{s.t. } & \alpha_j + \beta_i + \gamma_{ij} \leq (n+1-\sum_{\ell\in\mathcal{N}} \ell \cdot x_{j\ell})p_i && \forall i,j \in \mathcal{N} : i<j\\
& \alpha_j + \beta_i - \gamma_{ji} \leq (n+1-\sum_{\ell\in\mathcal{N}} \ell \cdot x_{j\ell})p_i && \forall i,j \in \mathcal{N} : i>j\\
& \alpha_i + \beta_i + \tau \leq (n+1-\sum_{\ell\in\mathcal{N}} \ell \cdot x_{j\ell})p_i && \forall i \in \mathcal{N}\\
& \sum_{i\in\mathcal{N}} a_{mi}p_i \leq b_m && \forall m \in \mathcal{M}\\
& p_i \geq 0 && \forall i \in \mathcal{N}\\
& \tau \geq 0.
\end{aligned}$$

Finally, we dualise this adversarial formulation to derive the following formulation for the recoverable problem:

$$\min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{q}} \sum_{m\in\mathcal{M}} b_m q_m \tag{7.44}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{N}} y_{ij} = 1 \qquad \forall j \in \mathcal{N} \tag{7.45}$$

$$\sum_{j \in \mathcal{N}} y_{ij} = 1 \qquad \forall i \in \mathcal{N} \tag{7.46}$$

$$\sum_{i \in \mathcal{N}} y_{ii} \geq n - 2\Delta \tag{7.47}$$

$$y_{ij} = y_{ji} \qquad \forall i, j \in \mathcal{N} \tag{7.48}$$

$$\sum_{m \in \mathcal{M}} a_{mi} q_m \geq \sum_{j \in \mathcal{N}} (n + 1 - \sum_{\ell \in \mathcal{N}} \ell \cdot x_{j\ell}) y_{ij} \qquad \forall i \in \mathcal{N} \tag{7.49}$$

$$\boldsymbol{x} \in \mathcal{X} \tag{7.50}$$

$$q_m \geq 0 \qquad \forall m \in \mathcal{M} \tag{7.51}$$

$$y_{ij} \geq 0 \qquad \forall i, j \in \mathcal{N} \tag{7.52}$$

As before, products $x_{j\ell} \cdot y_{ij}$ can be linearised using standard techniques. The resulting mixed-integer linear program contains $O(n^3)$ constraints and variables and can be found in Appendix C.2.3.

7.5 Comparison of formulations

This section presents a brief investigation into the linear relaxations of the three formulations derived above in order to compare their relative theoretical strengths. We begin by showing that no comparisons can be made between the general formulation and the other two formulations.

In preparation of the proof of this result we introduce budgeted uncertainty as a special case of polyhedral uncertainty. A budgeted uncertainty set can be defined as

$$\mathcal{U}_B = \left\{ \boldsymbol{p} \in \mathbb{R}^n_+ : \sum_{i \in \mathcal{N}} \frac{p_i - \hat{p}_i}{\bar{p}_i} \leq \Gamma,\, p_i \in [\hat{p}_i, \hat{p}_i + \bar{p}_i],\, i \in \mathcal{N} \right\},$$

where $\hat{p}_i$ is the nominal processing time of job i and $\bar{p}_i$ is the worst-case delay to the processing time of job i. Introduced by Bertsimas and Sim (2004), its motivation is to exclude unrealistically pessimistic worst-case scenarios from the uncertainty set and thereby avoid overly conservative and highly-expensive solutions. This is achieved by assuming that at most Γ jobs can simultaneously reach their maximum delays. Note that when $\Gamma = 0$, each job assumes its nominal processing time and the $\mathcal{U}_B$ reduces to a single scenario. Additionally, observe that as $\Gamma \to n$, this budgeted uncertainty set becomes an interval. When $\Gamma = n$ the worst-case scenario is known a priori to be when all jobs achieve their worst-case processing times $\hat{p}_i + \bar{p}_i$. In this case the problem can be solved by simply ordering the jobs according to their worst-case processing times, and no recourse action will be required. The proof of the following proposition makes use of an instance involving a budgeted uncertainty set.

Theorem 7.7. *The general formulation (C.5-C.25) is incomparable with both the matching-based formulation (C.26-C.42) and the assignment-based formulation (C.43-C.57).*

Proof. First consider a problem with two jobs with processing times that lie in the uncertainty set $\mathcal{U} = \{(p_1, p_2) : p_1 \leq 3,\, p_2 \leq 3,\, p_1 + 2p_2 \leq 7\}$. Suppose also that $\Delta = 1$, i.e. one swap can be made to amend the first-stage schedule. In this case, the linear relaxation of the matching-based formulation has an objective value of 7, whilst the linear relaxation of the assignment-based formulation has an objective value of 5.

Now consider an instance involving jobs with $\hat{\boldsymbol{p}} = (10, 8, 9, 4, 1, 5, 7, 1)$ and $\bar{\boldsymbol{p}} = (9, 7, 5, 4, 1, 3, 6, 1)$ lying in the budgeted uncertainty set $\mathcal{U}_B$, and set $\Gamma = 1$ and $\Delta = 1$. The linear relaxation of the matching-based formulation for this instance has an optimal objective value of -9.2 (to 1 decimal place), whilst the linear relaxation of the assignment-based formulation has an objective value of -285.4 (to 1 decimal place).

For both of these instances, the linear relaxation of the general formulation attains an objective value of 0. (In fact, for any polyhedral uncertainty set in which $a_{mi} \geq 0$ and $b_m \geq 0$ for all $i \in \mathcal{N}$, the linear relaxation of the general formulation will be 0, since it is free to set $h^k_{ii'j} = 0$ for all i, i', $j \in \mathcal{N}$, $k \in \mathcal{K}$ and therefore $q_m = 0$ for all $m \in \mathcal{M}$.)

These examples show that the matching and assignment-based formulations are tighter than the general formulation for some instances, but less tight for other instances. Hence the general formulation is incomparable with the matching and assignment-based formulations. $\square$

It is the case however that the objective value of the linear relaxation of the non-linear matching-based formulation is always greater than or equal to the objective value for the linear relaxation of the non-linear assignment-based formulation. That is, the non-linear matching formulation dominates the non-linear assignment formulation.

Theorem 7.8. *The non-linear matching-based formulation (7.28)-(7.34) dominates the non-linear assignment-based formulation (7.44)-(7.52).*

The proof of this statement involves the construction of a transformation ϕ to show that any feasible solution to the matching formulation can be transformed into a feasible solution to the assignment problem. The proof can be found in Appendix C.1.2. It does however remain open as to whether this result can be extended to the linearised versions of these formulations given by (C.26)-(C.42) and (C.43)-(C.57), respectively.

7.6 Computational experiments

This section presents and compares results from solving the three compact models introduced in this paper, as well as three additional heuristic solution methods. As a particular example of a general polyhedral uncertainty, here we consider budgeted uncertainty as outlined in the previous section. Before introducing three heuristics for solving this problem and examining their performance, we comment on the test instances and computational hardware used for these experiments.

Instances have been generated by randomly sampling both $\hat{p}_i$ and $\bar{p}_i$ from the set $\{1, 2, \ldots, 100\}$. 20 instances of sizes $n \in \{10, 15, 20\}$ have been generated, resulting in a total of 60 deterministic test instances. For each deterministic instance, three uncertain instances have been generated by setting $\Gamma \in \{3, 5, 7\}$, resulting in a total of 180 uncertain instances. These instances, as well as the complete results data, can be found at `https://github.com/boldm1/RR-single-machine-scheduling`.

All methods have been run on 4 cores of a 2.30GHz Intel Xeon CPU, limited to 16GB RAM. The exact models have been solved using Gurobi 9.0.1, with a time limit of 10 minutes.

7.6.1 Heuristics

The three heuristic methods we consider are as follows:

1. **Sorting**. Obtain a schedule by ordering the jobs $i \in \mathcal{N}$ according to non-decreasing $\hat{p}_i + \bar{p}_i$, i.e. a schedule that performs best in the worst-case scenario when $\Gamma = n$, and evaluate by solving $\text{Adv}(\boldsymbol{x})$.

2. **Max-min**. Solve the max-min problem

$$\max_{\boldsymbol{p} \in \mathcal{U}_B} \min_{\boldsymbol{x} \in \mathcal{X}} \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} p_i (n + 1 - j) x_{ij}$$

   to obtain a worst-case scenario $\boldsymbol{p} \in \mathcal{U}_B$. Find a schedule $\boldsymbol{x}$ that performs best in this worst-case scenario and evaluate by solving $\text{Adv}(\boldsymbol{x})$.

3. **Min-max**. Solve the problem without recourse, i.e. with $\Delta = 0$.

Each heuristic method has been used to find a feasible solution to all 180 uncertain test instances. Figure 7.3 shows the cumulative percentage of instances solved by each of the heuristics to within a given gap to the best solution found by any method, including the exact models, which have been solved with $\Delta = 2$. It is clear from this plot that min-max is the strongest of the three proposed heuristics, solving all 180 instances to within 3.2% of the best solution. This gap increases to 8.8% for sorting, whilst max-min solves all but one instance to within 15%. The averages of these gaps across all instances for min-max, sorting and max-min are 0.9%, 3.1% and 4.1% respectively.

Given its strong performance, we propose using min-max to provide a warm-start solution to the exact models. The benefits of this are assessed in the next section.

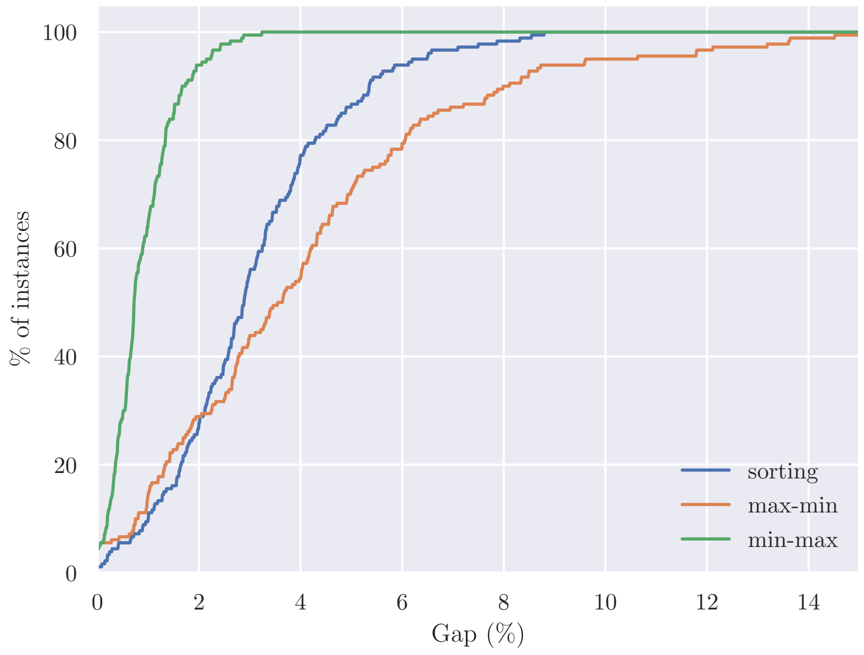


Figure 7.3: Cumulative percentage of instances solved to within a given gap of the best known solution.

7.6.2 Exact models

We now examine the results of solving the three exact models proposed in this paper and their warm-start variants. The 180 uncertain instances have been solved by each model and its warm-start variant for $\Delta \in \{0, 1, 2, 3\}$. Note that the general model has been implemented with $K = 2$. This has been chosen to make the general model as computationally efficient to solve as possible, whilst actually still providing an advantage over the min-max model, i.e. for $K = 1$ the general model corresponds to the min-max model.

Tables 7.1 and 7.2 compare the performance of these exact models for different values of Γ and Δ respectively. For each set of 20 instances with the same combination of instance parameters, Tables 7.1 and 7.2 report the following:

- *time* - Average CPU time (secs) required to solve the instances that were solved to optimality within the time limit.

- *LBgap* - Average gap (%) between the best objective bound and the best known feasible solution found by any method, over the instances not solved to optimality within the time limit.

- *UBgap* - Average gap (%) between the best feasible solution found within the time limit and the best known feasible solution found by any method, over the instances not solved to optimality within the time limit.

- *#solv* - Number of instances solved to optimality within the time limit.

From Tables 7.1 and 7.2, it is clear that the general model is by far the weakest of the three proposed models. Other than for $\Delta = 0$, no instances are solved to optimality. The general model is able to find near-optimal feasible solutions, but fails to begin closing the optimality gap in most instances. The matching-based model improves considerably on the general model, whilst the assignment model is the strongest performing of the three exact models, solving the most number of instances to optimality and having the smallest gaps over those instances that cannot be solved to optimality. The addition of a warm-start solution is clearly beneficial only for the assignment-based model, where the addition solves more instances to optimality in less time.

From Table 7.1 it can be seen that instances tend to become harder to solve as Γ increases from 3 to 7. From Table 7.2 we observe that, unsurprisingly, instances are easiest to solve to solve when $\Delta = 0$ (this corresponds to solving the min-max model). Interestingly however, when $n = 15$ and $n = 20$, instances are most difficult when $\Delta = 1$, and become easier to solve as the number of recovery swaps allowed, Δ, increases, i.e. the second stage-solution becomes less constrained by the first-stage solution.

Figure 7.4 shows performance profiles (Dolan and Moré, 2002) of the relative solution times of the matching and assignment-based models and their warm-start variants, for different instance sizes. The general model and its warm-start variant is excluded from these plots given its poor performance. A performance profile is a graphical comparison of the *performance ratios*. The performance ratio of model $m \in \mathcal{M}$ for instance $i \in \mathcal{I}$ is defined as

$$p_{im} = \frac{t_{im}}{\min_{m \in \mathcal{M}} t_{im}},$$

where t_{im} is the time required to solve instance i using model m. If model m fails to find an optimal solution to instance i within the given time-limit, then $p_{im} = P$, for some $P > \max_{i,m} r_{im}$. The performance profile of model $m \in \mathcal{M}$ is then defined to be the function

$$\rho_m(\tau) = \frac{|\{p_{im} \leq \tau : i \in \mathcal{I}\}|}{|\mathcal{I}|},$$

that is, the probability that model m is within a factor τ of the best performing model. The performance profiles in Figure 7.4 have been plotted on the log-scale for clarity.

The top-left performance profile in Figure 7.4 includes data from all instances, whilst the three other performance profiles consider the three sizes of instance separately. We see that for $n = 10$, the matching-based model performs slightly

			General				General + warm-start			
n	Γ	Δ	*time*	*LBgap*	*UBgap*	*#solv*	*time*	*LBgap*	*UBgap*	*#solv*
10	3	2	-	100.0	0.2	0	-	100.0	0.2	0
10	5	2	-	100.0	0.1	0	-	100.0	0.1	0
10	7	2	-	100.0	0.1	0	-	100.0	0.0	0
15	3	2	-	100.0	0.4	0	-	100.0	0.3	0
15	5	2	-	100.0	0.6	0	-	100.0	0.3	0
15	7	2	-	100.0	0.4	0	-	100.0	0.3	0
20	3	2	-	100.0	0.8	0	-	100.0	0.3	0
20	5	2	-	100.0	0.9	0	-	100.0	0.4	0
20	7	2	-	100.0	1.2	0	-	100.0	0.5	0
						0				0
			Matching				Matching + warm-start			
n	Γ	Δ	*time*	*LBgap*	*UBgap*	*#solv*	*time*	*LBgap*	*UBgap*	*#solv*
10	3	2	2.6	-	-	20	2.6	-	-	20
10	5	2	4.1	-	-	20	4.1	-	-	20
10	7	2	2.1	-	-	20	3.4	-	-	20
15	3	2	110.9	0.0	0.0	19	110.0	-	-	20
15	5	2	132.4	0.2	0.0	17	110.0	0.1	0.0	18
15	7	2	111.7	0.2	0.0	16	97.6	0.2	0.0	16
20	3	2	540.7	0.9	0.0	3	463.7	0.9	0.0	2
20	5	2	598.6	0.5	0.0	1	-	0.6	0.0	0
20	7	2	-	0.5	0.0	0	-	0.4	0.0	0
						116				116
			Assignment				Assignment + warm-start			
n	Γ	Δ	*time*	*LBgap*	*UBgap*	*#solv*	*time*	*LBgap*	*UBgap*	*#solv*
10	3	2	8	-	-	20	8.2	-	-	20
10	5	2	6.8	-	-	20	5.1	-	-	20
10	7	2	3.4	-	-	20	3.0	-	-	20
15	3	2	50.9	-	-	20	55.4	-	-	20
15	5	2	62.6	-	-	20	59.3	-	-	20
15	7	2	59.1	-	-	20	64.3	-	-	20
20	3	2	344.0	-	-	20	223.7	0.9	0	19
20	5	2	377.7	-	-	20	276.0	0.0	0	19
20	7	2	445.5	0.0	0.0	19	321.3	-	-	20
						179				178

Table 7.1: Comparison of the three exact models proposed in this paper and their warm-start variants, for different values of Γ.

			General				General + warm-start			
n	Γ	Δ	*time*	*LBgap*	*UBgap*	*#solv*	*time*	*LBgap*	*UBgap*	*#solv*
10	7	0	0.3	-	-	20	0.3	-	-	20
10	7	1	-	100.0	0.0	0	-	100.0	0.0	0
10	7	2	-	100.0	0.1	0	-	100.0	0.0	0
10	7	3	-	100.0	0.0	0	-	100.0	0.0	0
15	7	0	3.3	-	-	20	3.0	-	-	20
15	7	1	-	100.0	0.4	0	-	100.0	0.3	0
15	7	2	-	100.0	0.4	0	-	100.0	0.3	0
15	7	3	-	100.0	0.5	0	-	100.0	0.3	0
20	7	0	69.5	0.8	0.0	18	81.1	0.9	0.0	18
20	7	1	-	100.0	1.1	0	-	100.0	0.7	0
20	7	2	-	100.0	1.2	0	-	100.0	0.5	0
20	7	3	-	100.0	1.6	0	-	100.0	0.5	0
						58				58
			Matching				Matching + warm-start			
n	Γ	Δ	*time*	*LBgap*	*UBgap*	*#solv*	*time*	*LBgap*	*UBgap*	*#solv*
10	7	0	0	-	-	20	0.0	-	-	20
10	7	1	2.1	-	-	20	2.2	-	-	20
10	7	2	2.1	-	-	20	3.4	-	-	20
10	7	3	5.3	-	-	20	5.9	-	-	20
15	7	0	0.2	-	-	20	0.2	-	-	20
15	7	1	207	0.2	0.0	11	162.7	0.2	0.0	12
15	7	2	111.7	0.2	0.0	16	97.6	0.2	0.0	16
15	7	3	66.1	0.2	0.0	17	72.7	0.2	0.0	16
20	7	0	1.6	-	-	20	1.6	-	-	20
20	7	1	-	0.9	0.0	0	-	0.9	0.0	0
20	7	2	-	0.5	0.0	0	-	0.5	0.0	0
20	7	3	395.3	0.3	0.0	8	490.9	0.3	0.0	8
						172				172
			Assignment				Assignment + warm-start			
n	Γ	Δ	*time*	*LBgap*	*UBgap*	*#solv*	*time*	*LBgap*	*UBgap*	*#solv*
10	7	0	0.0	-	-	20	0.0	-	-	20
10	7	1	2.8	-	-	20	2.0	-	-	20
10	7	2	3.4	-	-	20	3.0	-	-	20
10	7	3	5.7	-	-	20	4.1	-	-	20
15	7	0	0.2	-	-	20	0.1	-	-	20
15	7	1	71.1	0.0	0.0	19	86.8	-	-	20
15	7	2	59.1	-	-	20	64.3	-	-	20
15	7	3	42.4	-	-	20	50.3	-	-	20
20	7	0	1.5	-	-	20	1.3	-	-	20
20	7	1	433.5	0.4	0.0	2	523.5	0.3	0.0	6
20	7	2	445.5	0.0	0.0	19	321.3	-	-	20
20	7	3	435.2	0.3	0.0	19	312.0	-	-	20
						219				226

Table 7.2: Comparison of the three exact models proposed in this paper and their warm-start variants, for different values of Δ.

better than the assignment-based model, however the inclusion of a warm-start does not seem to improve the matching model. For $n = 15$ and $n = 20$ however, the assignment model is stronger than the matching model. The benefits of a warm-start solution become most apparent when solving the largest instances, where a warm-start increases both solution times and the number of instances solved to optimality of both the matching and assignment model.

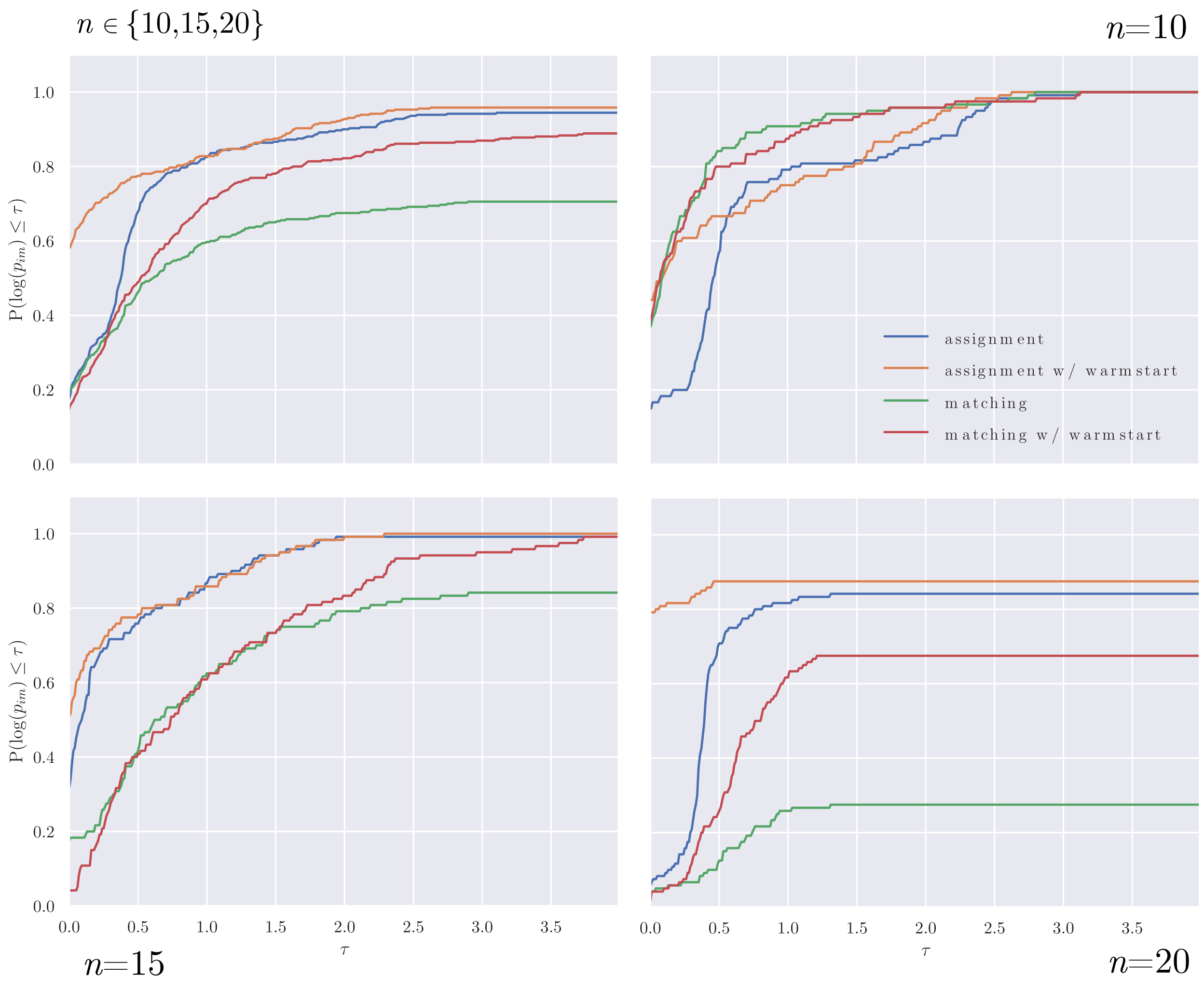


Figure 7.4: Performance profiles of relative solution times for different instance sizes.

7.6.3 Model parameters

We now examine the impact of the model parameters Γ and Δ on the objective value. For each set of instances, Tables 7.3 and 7.4 report the average objective value of the best known feasible solutions found by any method for different values of Γ and Δ respectively, as well as the relative percentage difference in this average from the sets of instances where $\Gamma = 3$ and $\Delta = 0$, respectively.

n	Γ	Δ	*avg. best*	*%diff.*
10	3	2	3946.5	0.0
10	5	2	4578.0	14.1
10	7	2	5053.0	22.1
15	3	2	7164.9	0.0
15	5	2	8177.5	12.7
15	7	2	9002.1	20.7
20	3	2	11814.3	0.0
20	5	2	13317.8	11.5
20	7	2	14582.5	19.3

Table 7.3: The effects of increasing Γ on the average objective value of the best known solution.

n	Γ	Δ	*avg. best*	*%diff.*
10	7	0	5079.7	0.0
10	7	1	5053.0	-0.5
10	7	2	5053.0	-0.5
10	7	3	5053.0	-0.5
15	7	0	9093.8	0.0
15	7	1	9002.2	-1.0
15	7	2	9002.1	-1.0
15	7	3	9002.1	-1.0
20	7	0	14735.6	0.0
20	7	1	14583.7	-1.0
20	7	2	14582.5	-1.0
20	7	3	14582.5	-1.0

Table 7.4: The effects of increasing Δ on the average objective value of the best known solution.

The results in Table 7.3 show that, as we would expect, increasing the Γ increases the average objective value in a concave manner. Table 7.4 shows that the inclusion of a second-stage recourse solution provides an improvement in objective value. However we also see that beyond $\Delta = 1$, increasing Δ provides little additional benefit. That is, the vast majority of the benefit of allowing a recourse solution can be captured by allowing just a single swap to the first-stage schedule. However, it is important to note the effect of having been limited to instances sizes of 20 and less by the computational intensity of solving the proposed exact models. We expect that for larger instance sizes, a less restricted and more powerful recourse action, i.e. increasing Δ, would become more advantageous. Additionally, for a discrete budgeted uncertainty set where $p_i \in \{\hat{p}_i, \hat{p}_i + \bar{p}_i\}$ for each $i \in \mathcal{N}$, we might expect the benefits of increasing Δ to be more apparent, since in this case the adversary is unable to spread the delay across multiple jobs in an attempt to preempt the recourse response, as is currently the case under the continuous budgeted uncertainty set that we consider. The impact of discrete budgeted uncertainty is an interesting possibility for future research on this problem.

7.7 Conclusions

This paper has introduced a recoverable robust model for the single machine scheduling problem with the total flow time criterion. A general result that allows for the construction of compact formulations for a wide range of recoverable robust problems has been presented, and this approach has been applied to the specific scheduling problem we consider. We have analysed the incremental subproblem of the robust scheduling problem in detail in an attempt to develop more tailored and effective compact formulations for this problem. Specifically, we have proved that matching problems with edge weights of the form of (7.21) have integral solutions, and therefore the inclusion of the odd-cycle constraints of the standard matching polytope is unnecessary. This result allows us to derive a matching-based compact formulation for the full recoverable robust single machine scheduling problem. A symmetric assignment-based formulation has also been presented, and we show how the integral matching result can be transferred to this alternative formulation to enable the derivation of a third compact model for this problem. Computational results show that this assignment-based model is the strongest of the three exact models.

There remain a number of promising directions in which future research on this problem can develop. Firstly, in this work we have considered a limited recourse action of allowing Δ disjoint swaps to be made to the first-stage schedule. Other measures of distance between the first and second-stage solution are certainly possible and worth investigating, especially if the restriction that the swapped pairs be disjoint could be relaxed, and interchanges between the positions of three or more jobs simultaneously can be factored into a recourse action. Another obvious avenue for future research is the analysis of this problem in the context of uncertainty sets different from polyhedral and budgeted uncertainty. Given the vast number of different objectives that have been used for single-machine scheduling problems and the unique properties of each, it would be interesting and worthwhile to investigate the application of this recoverable robust model to some of these. As a final suggestion, given the limited size of instance that have been solved by the exact models we propose, an accurate and effective heuristic approach for solving large-scale instances of this problem would certainly be a valuable development.

Chapter 8

Investigating the Recoverable Robust Single Machine Scheduling Problem Under Interval Uncertainty

8.1 Introduction

In this chapter, we consider a recoverable robust version of a single machine scheduling problem in which n jobs must be scheduled on a single machine without preemption, such that the total flow time, i.e. the sum of job completion times, is minimised. Under the $\alpha|\beta|\gamma$ scheduling notation introduced by Graham et al. (1979), the nominal problem is denoted as $1||\sum C_i$. In reality, job processing times are usually subject to some degree of uncertainty. When this is the case, it is important to account for this uncertainty when constructing solution schedules. Here, we consider the case where the job processing times are assumed to lie within specified intervals and examine the recoverable robust optimisation problem that arises from assuming a two-stage decision process.

The nominal problem can be stated as follows. Given a set of jobs $N = \{1, \ldots, n\}$ with processing times $\boldsymbol{p} = (p_1, \ldots, p_n)$, find an ordering of the jobs $j \in N$ such that the sum of completion times is minimised. In other words, we want to find a permutation σ of the set N, such that $\sum_{i \in N}(n+1-i)p_{\sigma(i)}$ is minimised. This nominal problem is easy to solve using the shortest processing time (SPT) rule of sorting the jobs by non-decreasing processing times. This problem can also be modelled as the following assignment problem with non-general cost structure:

$$\min \sum_{i \in N} \sum_{j \in N} p_j(n+1-i)x_{ij} \tag{8.1}$$

$$\text{s.t.} \sum_{i \in N} x_{ij} = 1 \qquad \forall j \in N \tag{8.2}$$

$$\sum_{j \in N} x_{ij} = 1 \qquad \forall i \in N \tag{8.3}$$

$$x_{ij} \in \{0,1\} \qquad \forall i, j \in N, \tag{8.4}$$

where $x_{ij} = 1$ if job j is scheduled in position i, and $x_{ij} = 0$ otherwise.

In this chapter we consider a two-stage decision process. Suppose that in the first-stage, the jobs $j \in N$ have processing times given by $\boldsymbol{p} = (p_1, \ldots, p_n)$, and in the second-stage they have different processing times given by $\boldsymbol{q} = (q_1, \ldots, q_n)$. We aim to find a first-stage schedule and second-stage schedule with minimum combined cost, such that the position of at least Δ jobs remain unchanged between the two schedules, i.e. only up to $n - \Delta$ jobs change position. This problem can be written as

$$\min \sum_{i \in N} \sum_{j \in N} p_j(n+1-i)x_{ij} + \sum_{i \in N} \sum_{j \in N} q_j(n+1-i)y_{ij} \tag{RecSMSP}$$

$$\text{s.t. } |X \cap Y| \geq \Delta$$

$$\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$$

where $X = \{(i, j) \in N \times N : x_{ij} = 1\}$ is the assignment corresponding to the first-stage schedule, $Y = \{(i, j) \in N \times N : y_{ij} = 1\}$ is the assignment corresponding to the second-stage schedule, and $\mathcal{X} = \{\boldsymbol{x} \in \{0, 1\}^{n \times n} : (8.2), (8.3)\}$ is the set of feasible schedules. The condition $|X \cap Y| \geq \Delta$ is equivalent to demanding that

$$\sum_{i \in N} \sum_{j \in N} x_{ij} y_{ij} \geq \Delta.$$

(RecSMSP) can be modelled as a mixed-integer program (MIP) following the linearisation of this constraint. This is achieved with the introduction of an additional set of $\boldsymbol{z}$ variables, and the following constraints:

$$z_{ij} \leq x_{ij} \qquad \forall i, j \in N$$

$$z_{ij} \leq y_{ij} \qquad \forall i, j \in N$$

$$\sum_{i \in N} \sum_{j \in N} z_{ij} \geq \Delta$$

$$z_{ij} \in \{0, 1\} \qquad \forall i, j \in N$$

Observe how the $\boldsymbol{z}$ variables represent the shared assignments of the first and second-stages, i.e. $z_{ij} = 1$ if job j is assigned to position i in both the first and second-stage schedules.

This problem can be considered to be a recoverable robust optimisation problem with interval uncertainty (see Liebchen et al. (2009)). A recoverable robust optimisation problem consists of two-stages. A full first-stage solution must be determined under the problem uncertainty, before an adversary chooses a worst-case realisation of the uncertain data. Then, in response to this realisation, the first-stage solution can be recovered, i.e. amended in some limited way, to obtain a second-stage solution. For a general survey on robust discrete optimisation problems, we refer the reader to Kasperski and Zieliński (2016a).

We consider exactly this recoverable robust problem in the setting where the second-stage job processing times $\boldsymbol{q}$, are uncertain, but known to lie within an interval (or box) uncertainty set given by

$$\mathcal{U} = \left\{ \boldsymbol{q} \in \mathbb{R}^n_+ : q_j \in [\hat{q}_j - \bar{q}_j, \hat{q}_j + \bar{q}_j],\ j \in N \right\}.$$

That is, each job $j \in N$ has a nominal second-stage processing time given by $\hat{q}_j$, but can deviate from this nominal value by up to $\bar{q}_j$. The worst-case scenario for this uncertainty set occurs when each job simultaneously achieves its worst-case processing time, i.e. $\hat{q}_j + \bar{q}_j$ for each $j \in N$. Denoting the first-stage duration of $j \in N$ as p_j, and its worst-case second-stage duration as q_j, we get (RecSMSP).

There exists a number of papers that consider various discrete optimisation problems in the setting of recoverable robust optimisation with interval uncertainty. The resulting problems they study therefore also contain the same intersection constraints that we consider here. Our work in this chapter extends this framework to the context of single-machine scheduling. Büsing (2012) studies recoverable robust shortest path problems, whilst Hradovich et al. (2017a,b) investigate the recoverable robust spanning tree problem. Kasperski and Zieliński (2017) consider the recoverable robust selection problem under both discrete and interval uncertainty and, most recently, Goerigk et al. (2021b) consider the recoverable robust travelling salesman problem. The single-machine scheduling problem we consider is an assignment problem with a specific cost structure, and therefore Fischer et al. (2020) is the paper that is most closely related to our work. In this paper, the authors examine the complexity of the recoverable robust assignment problem with interval uncertainty. Amongst other results, they show that this problem is W[1]-hard with respect to Δ and $n - \Delta$. Even though they use only 0-1 costs for this reduction, note that this hardness result does not extend to the scheduling cost structure that we consider here.

As a brief comment on other papers that consider robust assignment problems, Deĭneko et al. (2006) study the problem in the context of discrete scenarios and Pereira and Averbakh (2011) considers interval uncertainty in combination with the regret criterion.

The single machine scheduling problem with the objective of minimising the (weighted) sum of completion times under uncertain job processing times is well-studied, particularly for the case of discrete uncertain scenarios. Daniels and Kouvelis (1995), Kouvelis and Yu (1997), Yang and Yu (2002) and Aloulou and Della Croce (2008) show that even for just two discrete scenarios, robust versions of this problem are NP-hard, whilst Mastrolilli et al. (2013) show that no polynomial-time approximation algorithm exists. Zhao et al. (2010) propose a cutting plane algorithm to solve the problem. More recently, Kasperski and Zieliński (2016b) considered the problem for the ordered weighted averaging (OWA) criterion, of which classical robustness is a special case, and Kasperski and Zieliński (2019) considered the problem for the value at risk (VaR) and conditional value at risk

(CVaR) criteria.

The case of interval uncertainty has also received a lot of attention. Daniels and Kouvelis (1995) characterised a set of dominance relations between jobs in an optimal schedule in the case of interval uncertainty. Lebedev and Averbakh (2006) showed that the problem is NP-hard for regret robustness, whilst Montemanni (2007) presented a compact MIP that was shown to be able to solve instances involving up to 45 jobs. Kasperski and Zieliński (2008) showed that the regret problem is 2-approximable when the corresponding nominal problem is solvable in polynomial time. For a survey of robust single machine scheduling for both discrete and interval uncertainty, see Kasperski and Zielinski (2014).

For robust single machine scheduling in the context of budgeted uncertainty, Lu et al. (2014) presented an MIP and a heuristic to solve the problem, before Bougeret et al. (2019) examined its complexity. The work presented in the previous chapter of this thesis (Bold and Goerigk, 2022b) considered the recoverable robust problem under a different similarity measure to the one considered in this chapter.

The structure and contributions of this chapter are as follows. Section 8.2 presents a number of positive results for the problem, including an efficient method for optimal scheduling when a set of jobs which must have the same first and second-stage positions is given. Section 8.3 then presents a 2-approximation algorithm and greedy heuristic for the full problem. The performance of the exact MIP formulation as well as the heuristics we present are examined experimentally in Section 8.4, before concluding remarks are made in Section 8.5.

8.2 Problem properties

We begin our analysis of the recoverable robust single machine scheduling problem by considering the following question: Given a set $M \subseteq N$ of jobs that must be scheduled in the same position in both the first and the second stage, what is the best possible solution? That is, we consider the problem of calculating

$$\begin{aligned} f(M) = \min & \sum_{i\in N}\sum_{j\in N} p_j(n+1-i)x_{ij} + \sum_{i\in N}\sum_{j\in N} q_j(n+1-i)y_{ij} && \text{(RecFix)} \\ \text{s.t. } & x_{ij} = y_{ij} && \forall i \in N, j \in M \\ & \boldsymbol{x}, \boldsymbol{y} \in \mathcal{X} \end{aligned}$$

Note that a solution $(\boldsymbol{x}, \boldsymbol{y})$ of (RecFix) may intersect on more jobs than just those in the set M. Also note that if $|M| \geq \Delta$, then the solution to this problem is feasible to (RecSMSP). Therefore, the optimal objective value of (RecSMSP) is equal to $\min\{f(M) : |M| \geq \Delta\}$. In Algorithm 8.1, we show how to calculate the value $f(M)$ in polynomial time.

The following example demonstrates the implementation of this ordering rule. Consider the data shown in Table 8.1 and suppose that $M = \{3, 4\}$, i.e. jobs 3 and 4 must share the same position in the first and second-stage schedules. Then $N \setminus M = \{1, 2, 5\}$, i.e., jobs 1, 2 and 5 can be assigned different positions in the two schedules. The sorted first and second-stage processing times of jobs in $N \setminus M$ are $\boldsymbol{a} = (2, 3, 5)$ and $\boldsymbol{b} = (1, 4, 6)$ respectively, and the joint processing times for the jobs in M are $\boldsymbol{c} = (14, 6)$. Then $\boldsymbol{d} = \boldsymbol{a} + \boldsymbol{b} = (3, 7, 11)$ and $\boldsymbol{e} = (3, 6, 7, 11, 14)$ as the merged and sorted vector processing times. This corresponds to the first-stage schedule $\sigma_1 = (5, 4, 2, 1, 3)$, and the second-stage schedule $\sigma_2 = (2, 4, 1, 5, 3)$. Note that, as required, jobs 3 and 4 are placed in the same position in both the first and second-stage schedules.

Algorithm 8.1 Evaluation method for $f(M)$

```
1: procedure EVAL($\boldsymbol{p}, \boldsymbol{q}, M$)
2:     Set $\boldsymbol{a} = (a_j)_{j \in N \setminus M}$ to be the vector of values $p_j$, $j \in N \setminus M$, sorted by non-decreasing values
3:     Set $\boldsymbol{b} = (b_j)_{j \in N \setminus M}$ to be the vector of values $q_j$, $j \in N \setminus M$, sorted by non-decreasing values
4:     Set $\boldsymbol{c} = (c_j)_{j \in M}$ to be the vector of values $p_j + q_j$, $j \in M$
5:     Set $\boldsymbol{d} = \boldsymbol{a} + \boldsymbol{b} = (a_1 + b_1, \ldots, a_{N \setminus M} + b_{N \setminus M})$
6:     Let $\boldsymbol{e} = (e_j)_{j \in N}$ be the vector found by concatenating vectors $\boldsymbol{c}$ and $\boldsymbol{d}$ and sorting by non-decreasing values
7:     return $\sum_{i \in N} (n + 1 - i) e_i$
8: end procedure
```

j	1	2	**3**	**4**	5
p_j	5	3	**5**	**1**	2
q_j	4	1	**9**	**5**	6
$p_j + q_j$	9	4	**14**	**6**	8

Table 8.1: Example problem data. Columns of $M = \{3, 4\}$ are in bold.

We now show that Algorithm 8.1 does indeed give an optimal solution to problem (RecFix).

Theorem 8.1. *Algorithm 8.1 calculates $f(M)$ for any $M \subseteq N$.*

Proof. Let any $M \subseteq N$ be given. We denote these jobs as $M = \{j_1, \ldots, j_m\}$ with $m = |M|$. Let us first assume that we already know the set of slots $K = \{i_1, \ldots, i_m\} \subseteq N$ into which the jobs will be scheduled. We will then show how to find such a set. We further denote by $N \setminus M = \{j'_1, \dots, j'_{n-m}\}$ and $N \setminus K = \{i'_1, \dots, i'_{n-m}\}$ the sets of jobs and slots that are not part of M and K, respectively. Without loss of generality, we assume that $i_1 < i_2 < \ldots < i_m$ and $i'_1 < i'_2 < \ldots < i'_{n-m}$. The resulting problem thus decomposes into the two subproblems

$$\min_{\boldsymbol{z} \in \mathcal{X}} \sum_{k=1}^{m} \sum_{\ell=1}^{m} (n+1-i_k)(p_{j_\ell} + q_{j_\ell}) z_{k\ell} + \min_{\boldsymbol{x},\boldsymbol{y} \in \mathcal{X}} \sum_{k=1}^{n-m} \sum_{\ell=1}^{n-m} (n+1-i'_k)(p_{j'_\ell} x_{k\ell} + q_{j'_\ell} y_{k\ell}),$$

where with slight abuse of notation, $\mathcal{X}$ denotes the set of assignments with suitable dimension. Note that independent of the choice of K, we can find optimal solutions $\boldsymbol{z}^*$, $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$. Sorting jobs $j \in M$ by non-decreasing joint processing times $p_j + q_j$ gives $\boldsymbol{z}^*$. We denote the resulting vector of joint processing times as $\boldsymbol{c} = (c_j)_{j \in M}$. Sorting jobs $j \in N \setminus M$ by non-decreasing p_j gives $\boldsymbol{x}^*$, and by non-decreasing q_j gives $\boldsymbol{y}^*$. We denote the resulting vector of sums of the sorted processing times as $\boldsymbol{d} = (d_j)_{j \in N \setminus M}$.

Let us fix these assignments accordingly and consider how to find set K. This is equivalent to assigning the positions $i_1, \dots, i_m$ and $i'_1, \dots, i'_{n-m}$ to slots in N. Hence, the value $f(M)$ is equal to

$$\min_{\boldsymbol{\tau} \in \mathcal{X}} \sum_{i \in N} \sum_{\ell=1}^{m} (n+1-i\tau_{i\ell}) c_\ell + \sum_{i \in N} \sum_{\ell=1}^{n-m} (n+1-i\tau_{i,\ell+m}) d_\ell$$

An optimal solution to this problem can be found by assigning the positions according to non-decreasing processing times of their associated jobs, i.e., to sort the concatenated vector of processing times consisting of $\boldsymbol{c}$ and $\boldsymbol{d}$. Algorithm 8.1 is exactly this solution procedure. □

Therefore, if it is known which set of jobs M with cardinality Δ must share a position in the first and second-stage schedule, the remaining problem can be solved in $O(n \log n)$ time. This immediately gives the following result.

Corollary 8.2. *For a constant value of Δ, (RecSMSP) can be solved in polynomial time $O(n^{\Delta+1} \log n)$.*

Proof. This can be seen by simply enumerating the $\binom{n}{\Delta} \sim O(n^{\Delta})$ possible ways of choosing a set of Δ jobs to share first and second-stage assignments. For each of the $O(n^{\Delta})$ ways of fixing Δ jobs, the remaining problem can be solved using Algorithm 8.1 in $O(n \log n)$ time. Hence, the overall complexity is given by $O(n^{\Delta+1} \log n)$, i.e. polynomial for fixed Δ. □

As observed in Fischer et al. (2020), it is straightforward to see that problem (RecSMSP) can be solved in polynominal time for constant value of Δ by enumerating all possibilities for the intersection set $|X \cap Y|$. Note that whilst this approach would require us to check $\binom{n^2}{\Delta}$ many candidates for general cost functions, this number is reduced to $\binom{n}{\Delta}$ in our case.

We now consider a special case of (RecSMSP) where there are a constant number k of possible job processing times, i.e. $p_j, q_j \in \{d_1, \ldots, d_k\}$, where $d_1, \ldots, d_k \in \mathbb{R}$ for constant k. Note that this results in at most k^2 possible combinations of first and second-stage costs p_j and q_j, that is k^2 possible job types. Furthermore, $f(M) = f(M')$ for any two choices $M, M' \subseteq N$ that contain the same number of each job type. We can conclude the following result.

Corollary 8.3. *Let $p_j, q_j \in \{d_1, \ldots, d_k\}$ for a constant value of k. Then, (RecSMSP) can be solved in polynomial time $O(n^{k^2})$*

Proof. We consider the number of ways there are to choose Δ jobs from k^2 different job types. Observe that for all but the final job type, there are $\Delta + 1$ ways of choosing up to Δ jobs of that type. Having chosen the number of jobs from the first k^2-1 job types, the number of jobs of the final type is simply equal to the number of jobs remaining. Hence there are $O((\Delta+1)^{k^2-1})$ ways to choose the Δ jobs. For each of these choices, we can use the ordering rule presented in Algorithm 8.1 to solve the resulting problem. Note that sorting n bounded values is possible in $O(n)$ time. Hence (RecSMSP) with k possible job processing times is solvable in $O((\Delta+1)^{k^2-1}\cdot n) \sim O(n^{k^2})$ time. □

8.3 A 2-approximation algorithm

In this section, we present an approximate solution to (RecSMSP) and prove that this solution has a guaranteed worst-case approximation ratio of 2.

Theorem 8.4. *A solution to the problem*

$$f(N) = \min \sum_{i\in N}\sum_{j\in N} p_j(n+1-i)x_{ij} + \sum_{i\in N}\sum_{j\in N} q_j(n+1-i)y_{ij} \qquad \text{(UB-SMSP)}$$

$$|X\cap Y| = n$$

$$\boldsymbol{x},\, \boldsymbol{y} \in \mathcal{X}$$

provides a 2-approximation to (RecSMSP).

Observe that (UB-SMSP) is equivalent to (RecSMSP) with complete intersection of the first and second-stage solutions. It can thus be simplified to

$$\min_{\boldsymbol{x}\in\mathcal{X}} \sum_{i\in N}\sum_{j\in N}(p_j+q_j)(n+1-i)x_{ij}$$

Note that (UB-SMSP) can be solved in $O(n\log n)$ time by simply ordering the jobs $j \in N$ according to non-decreasing $p_j + q_j$ values.

In the following, we provide a proof for Theorem 8.4. To this end, we consider a lower bound to (RecSMSP), which is provided by a solution to the problem

$$\left(\min_{\boldsymbol{x}\in\mathcal{X}} \sum_{i\in N}\sum_{j\in N} p_j(n+1-i)x_{ij}\right) + \left(\min_{\boldsymbol{y}\in\mathcal{X}} \sum_{i\in N}\sum_{j\in N} q_j(n+1-i)y_{ij}\right), \quad \text{(LB-SMSP)}$$

that is, (RecSMSP) without the intersection constraint. Suppose σ_p and σ_q are orderings of jobs $j \in N$ by non-decreasing first-stage costs p_j and non-decreasing second-stage costs q_j, respectively. Then (LB-SMSP) has objective value equal to

$$\sum_{j\in N}(p_{\sigma_p(j)} + q_{\sigma_q(j)})(n+1-j).$$

We compare the upper bound provided by (UB-SMSP) with the lower bound provided by (LB-SMSP) and refer to the values of these upper and lower bounds as UB and LB, respectively.

By sorting items by σ_p and σ_q and considering the different positions of specific items between these two sortings, we get a different perspective on what constitutes an instance of (RecSMSP). Specifically, we can consider sorted processing times $p_1 \leq p_2 \leq \ldots \leq p_n$ and $q_1 \leq q_2 \leq \ldots \leq q_n$, and a permutation π, such that the processing times p_j and $q_{\pi(j)}$ belong to the same job. Figure 8.1 provides an example of a (RecSMSP) considered in this way.

Using this description of an instance, we begin by first proving the following structural lemma, which shows that a worst-case instance is of a form where the smallest value of $\boldsymbol{p}$ is matched with the largest value of $\boldsymbol{q}$, the second smallest value of $\boldsymbol{p}$ is matched with the second-largest value of $\boldsymbol{q}$, and so on.

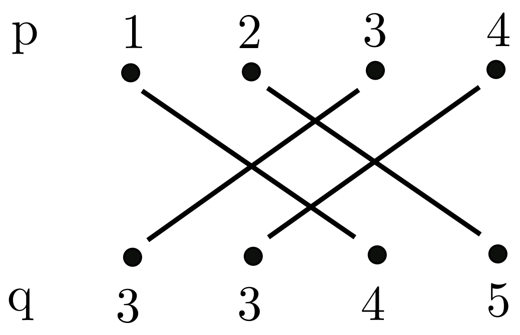


Figure 8.1: An example instance of (RecSMSP) involving four jobs. First-stage processing times are shown in the top row and second-stage processing times are shown in the bottom row. An edge between first-stage node and second-stage node indicates that these processing times belong to a single job.

Lemma 8.5. *Let any instance of (RecSMSP) be given, consisting of sorted vectors* $\boldsymbol{p}$, $\boldsymbol{q}$ *and a permutation* π. *Then the ratio* $\frac{UB}{LB}$ *for this instance is less or equal to the ratio for the instance where* π *is replaced by* π', *where* $\pi' = (n, n-1, \dots, 1)$, *i.e.,* π' *is a sorting of indices from largest to smallest. We refer to such an instance as being 'fully-crossed'.*

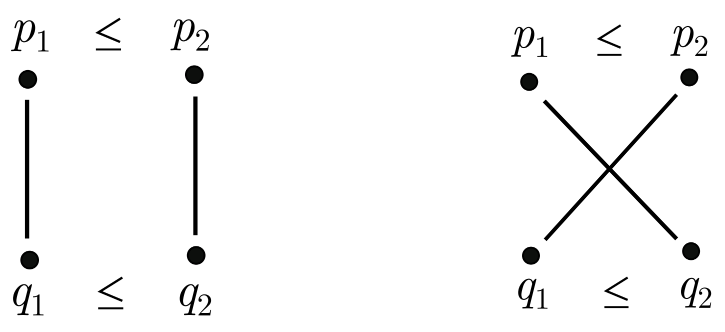
Proof. Matching a first-stage cost to a second-stage cost represents a job having those respective processing times. Here, we match first and second-stage costs with the objective of maximising the ratio $\frac{UB}{LB}$. To this end, observe that LB is the same for all possible matchings π as the intersection constraint is ignored, and hence finding a matching to maximise the ratio $\frac{UB}{LB}$ is equivalent to finding a matching that maximises UB.

Let us consider the dual problem of UB, given by

$$\max \sum_{i \in N} u_i + v_i$$

$$\text{s.t. } u_j + v_i \leq (n+1-i)(p_j + q_{\pi(j)}) \qquad \forall i, j \in N$$

Consider any two jobs, and suppose that first and second-stage costs are labelled so that $p_1 \leq p_2$ and $q_1 \leq q_2$. We compare the objective values between the case when p_1 is matched to q_1 and p_2 is matched to q_2, i.e. the two jobs are uncrossed (see Figure 8.2a), and the case when p_1 is matched to q_2 and p_2 is matched to q_1, i.e. the two jobs are crossed (see Figure 8.2b).



(a) Uncrossed matching. (b) Crossed matching.

Figure 8.2: Uncrossed and crossed instances.

Let $(\boldsymbol{u}^*, \boldsymbol{v}^*)$ be an optimal solution to the dual of (UB-SMSP) for the instance in which the two jobs are uncrossed, and let us denote its objective value as $UB_{\text{uncrossed}}$. Note that

$$u_1^* = \min_{i \in N} \{(n+1-i)(p_1+q_1) - v_i^*\}$$

$$u_2^* = \min_{i \in N} \{(n+1-i)(p_2+q_2) - v_i^*\}.$$

Now consider the dual of (UB-SMSP) for the instance in which the two jobs are crossed. We denote its objective value as UB_{crossed} and construct a feasible solution $(\boldsymbol{u}, \boldsymbol{v})$ by setting $\boldsymbol{v} = \boldsymbol{v}^*$,

$$u_1 = \min_{i \in N} \{(n+1-i)(p_1+q_2) - v_i^*\}$$

$$u_2 = \min_{i \in N} \{(n+1-i)(p_2+q_1) - v_i^*\}$$

and $u_j = u_j^*$ for all other jobs j. Since this is a feasible solution, we have $UB_{\text{crossed}} \geq \sum_{i \in N} u_i + v_i$.

Now for $\lambda \in [0,1]$, we consider the function $f_i(\lambda) = (n+1-i)(\lambda(p_1+q_1) + (1-\lambda)(p_2+q_2)) - v_i^*$ and define $g(\lambda) = \min_{i \in N} f_i(\lambda) + \min_{i \in N} f_i(1-\lambda)$. The minimum of concave functions is concave, and the sum of concave functions is concave, and therefore function g is concave. Furthermore, it is symmetric with respect to $\lambda = 0.5$. Hence, g is minimised for $\lambda = 0$ and $\lambda = 1$, and thus for all $\lambda \in [0,1]$, $g(\lambda) \geq g(0) = g(1)$.

We therefore conclude that

$$\begin{aligned} UB_{\text{crossed}} - UB_{\text{uncrossed}} &\geq \sum_{i \in N} u_i + v_i - \sum_{i \in N} u_i^* + v_i^* \\ &= u_1 + u_2 - u_1^* - u_2^* \\ &= g(\lambda) - g(0) \geq 0 \end{aligned}$$

where $\lambda = (p_2 - p_1)/(p_2 + q_2 - p_1 - q_1)$. Hence, by changing the matching π as described, the value of the upper bound is not decreased. Repeating this process, we find that there is always a fully-crossed worst-case instance as claimed. □

We are now in a position to prove Theorem 8.4.

Proof of Theorem 8.4. We suppose that the first and second-stage costs are ordered so that $p_1 \leq p_2 \leq \cdots \leq p_n$ and $q_1 \leq q_2 \leq \cdots \leq q_n$, and that the jobs are labelled in order of their first-stage costs. Making use of Lemma 8.5, we restrict our consideration to instances where the matching of first and second-stage costs is fully crossed, i.e. job j has the j-th largest first-stage cost p_j, and the $(n-j)$-th largest second-stage cost q_{n-j}.

We examine the problem of choosing values $\boldsymbol{p}$ and $\boldsymbol{q}$ to maximise $\frac{UB}{LB}$, i.e.

$$\max_{(\boldsymbol{p},\boldsymbol{q})\in\mathcal{PQ}} \min_{\pi\in\Pi} \frac{\sum_{i\in N}(n+1-i)(p_{\pi(i)}+q_{n-\pi(i)})}{\sum_{i\in N}(n+1-i)(p_i+q_i)}, \tag{8.5}$$

where $\pi \in \Pi$ is a permutation of the positions $i \in N$ such that $\pi(i)$ is the job scheduled in position i, and $\mathcal{PQ} = \{(\boldsymbol{p},\boldsymbol{q}) \in \mathbb{R}_+^{n+n} : p_1 \leq p_2 \leq \cdots \leq p_n,\ q_1 \leq q_2 \leq \cdots \leq q_n\}$.

Normalising the objective function we get

$$\max_{(\boldsymbol{p},\boldsymbol{q})\in\overline{\mathcal{PQ}}} \min_{\pi\in\Pi} \sum_{i\in N}(n+1-i)(p_{\pi(i)}+q_{n-\pi(i)}),$$

where $\overline{\mathcal{PQ}} = \{(\boldsymbol{p},\boldsymbol{q}) \in \mathcal{PQ} : \sum_{i\in N}(n+1-i)(p_i+q_i) = 1\}$. For fixed $\boldsymbol{p}$ and $\boldsymbol{q}$, the inner minimisation problem over the set of permutations $\pi \in \Pi$ can be written as the following assignment problem

$$\begin{aligned}
\min \ & \sum_{i\in N}\sum_{j\in N}(n+1-i)(p_j+q_{n-j})x_{ij} & \\
\text{s.t.} \ & \sum_{i\in N} x_{ij} = 1 & \forall j \in N \\
& \sum_{j\in N} x_{ij} = 1 & \forall i \in N \\
& x_{ij} \in \{0,1\} & \forall i,\ j \in N,
\end{aligned}$$

where $x_{ij} = 1$ indicates that job j is scheduled in position i, and $x_{ij} = 0$ otherwise. Note that the binary constraints on the x_{ij} variables can be relaxed to $x_{ij} \geq 0$ for all $i,\, j \in N$. Hence, taking the dual of this linear program, we get

$$\begin{aligned}
&\max \sum_{i\in N} u_i + v_i \\
&\text{s.t.}\; u_j + v_i \leq (n+1-i)(p_j + q_{n-j}) && \forall i,\, j \in N,
\end{aligned}$$

and therefore the full problem (8.5) can be written as

$$\begin{aligned}
\max &\sum_{i\in N} u_i + v_i \\
\text{s.t.} &\sum_{i\in N} (n+1-i)(p_i + q_i) = 1 && && (\gamma) \\
&u_j + v_i \leq (n+1-i)(p_j + q_{n-j}) && \forall i,\, j \in N && (\boldsymbol{x}) \\
&p_i \leq p_{i+1} && \forall j \in N \setminus \{n\} && (\boldsymbol{\alpha}) \\
&q_i \leq q_{i+1} && \forall j \in N \setminus \{n\} && (\boldsymbol{\beta}) \\
&p_i,\, q_i \geq 0 && \forall i \in N.
\end{aligned}$$

Dualising this problem (the corresponding dual variables are shown to the right of each of the above constraints) gives

$$\begin{aligned}
&\min \gamma \\
&\text{s.t.} \sum_{i\in N} x_{ij} = 1 && \forall j \in N && (\boldsymbol{u}) \\
&\quad\;\; \sum_{j\in N} x_{ij} = 1 && \forall i \in N && (\boldsymbol{v})
\end{aligned}$$

$$\left.\begin{aligned}
&n\gamma - \sum_{i\in N}(n+1-i)x_{i1} + \alpha_1 \geq 0 \\
&(n+1-j)\gamma - \sum_{i\in N}(n+1-i)x_{ij} - \alpha_j + \alpha_{j+1} \geq 0 \quad \forall j \in N \setminus \{1,n\} \\
&\gamma - \sum_{i\in N}(n+1-i)x_{in} - \alpha_n \geq 0
\end{aligned}\right\} \quad (\boldsymbol{p})$$

$$\left.\begin{aligned}
&n\gamma - \sum_{i\in N}(n+1-i)x_{in} + \beta_1 \geq 0 \\
&(n+1-j)\gamma - \sum_{i\in N}(n+1-i)x_{i,n+1-j} - \beta_j + \beta_{j+1} \geq 0 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad \forall j \in N \setminus \{1,n\} \\
&\gamma - \sum_{i\in N}(n+1-i)x_{i1} - \beta_n \geq 0
\end{aligned}\right\} \quad (\boldsymbol{q})$$

$$x_{ij} \geq 0 \qquad \forall i,\, j \in N$$

$$\alpha_i,\, \beta_i \geq 0 \qquad \forall i \in N$$

(again, the corresponding primal variables are shown to the right of each of the above constraints).

We now proceed to show that there is a feasible solution to this dual problem with an objective value of 2. Since a feasible solution to the dual problem provides an upper bound to the primal problem, finding a feasible dual solution with an objective value of 2 guarantees that the ratio $\frac{UB}{LB}$ is bounded above by 2, and therefore proves that (UB-SMSP) does indeed provide a 2-approximation to (RecSMSP).

Setting $\alpha_i = \beta_i = 0$ for all $i \in N$, the dual problem becomes

$$\min \gamma$$

$$\begin{aligned}
\text{s.t.}\ & \sum_{i\in N} x_{ij} = 1 && \forall j \in N \\
& \sum_{j\in N} x_{ij} = 1 && \forall i \in N \\
& (n+1-j)\gamma \geq \sum_{i\in N}(n+1-i)x_{ij} && \forall j \in N && (*) \\
& (n+1-j)\gamma \geq \sum_{i\in N}(n+1-i)x_{i,n+1-j} && \forall j \in N && (**) \\
& x_{ij} \geq 0 && \forall i,\, j \in N.
\end{aligned}$$

Observe that constraint (**) can be rewritten as

$$j\gamma \geq \sum_{i\in N}(n+1-i)x_{ij} \quad \forall j \in N.$$

Suppose that $\gamma = 2$. Then constraints (*) and (**) enforce that

$$\begin{aligned}
2(n+1-j) &\geq n+1-\sum_{i\in N} i x_{ij} && \forall j \in N \\
2j &\geq n+1-\sum_{i\in N} i x_{ij} && \forall j \in N,
\end{aligned}$$

that is

$$\sum_{i\in N} i x_{ij} \geq \max\{2j-n-1,\, n+1-2j\} \quad \forall j \in N \qquad (\dagger)$$

By considering the above constraint, we can determine an approach for generating a feasible dual solution with objective value $\gamma = 2$. This approach works as follows: take the positions $i = n, n-1, \ldots, 1$ in decreasing order, and assign jobs $j \in N$ to them, alternating between the unassigned job with the largest index and the unassigned job with the smallest index.

We illustrate this approach with an example for $n = 6$. Table 8.2 shows

j	1	2	3	4	5	6
$\max\{2j-n-1,\, n+1-2j\}$	5	3	1	1	3	5

Table 8.2: Right-hand side of constraint (†) for each $j \in N$ when $n = 6$.

constraint (†) for each $j \in N$. $\sum_{i\in N} ix_{ij}$ is the position in the schedule that job j is assigned to, and hence these constraints enforce that jobs 1 and 6 are scheduled in the final two positions, jobs 2 and 5 are scheduled in the two positions before that, and jobs 3 and 4 are scheduled in the first two positions. Thus there are two feasible positions for each job. Following the proposed approach for generating a feasible solution, we get that $x_{6,6} = 1$, $x_{5,1} = 1$, $x_{4,5} = 1$, $x_{3,2} = 1$, $x_{2,4} = 1$, $x_{1,3} = 1$.

For any given instance, this approach can be used to generate a UB solution with objective value no worse than twice the LB solution provided by (LB-SMSP).

$\square$

We next consider the value of $\frac{UB}{LB}$ as a function of n and show that the 2-approximation provided by the solution (UB-SMSP) becomes tight as $n \to \infty$.

First, consider the case in which n is even. We construct a fully-crossed instance with $p_i = q_i = 0$ for $i \leq n/2$ and $p_i = q_i = 1$ for $i \geq n/2 + 1$ (which we refer to as a fully-crossed 0-1 instance). Such an instance for $n = 4$ is shown in Figure 8.3a. The value of UB is given by $\sum_{i\in N}(n + i - 1) = \frac{(n+1)n}{2}$, whilst the value of LB is given by

$$\sum_{i\in N,\, i\geq \frac{n}{2}} 2(n+1-i) = \frac{2(n/2+1)n/2}{2} = \frac{(n/2+1)n}{2},$$

and hence when n is even, we have that

$$\frac{UB}{LB} = \frac{n+1}{n/2+1} = \frac{2n+2}{n+2} \leq 2.$$

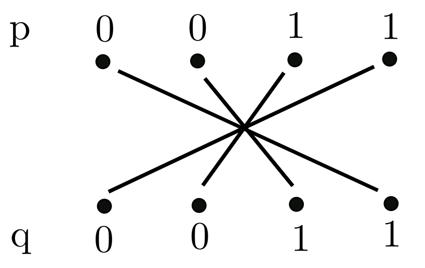
Now consider the case in which n is odd. We slightly change the construction of this instance by setting $p_i = q_i = 0$ for $i \leq (n-1)/2$ and $p_i = q_i = 1$ for $i \geq (n+1)/2 + 1$ and $p_i = 0$, $q_i = 1$ for $i = (n+1)/2$. This instance, for $n = 5$, is shown in Figure 8.3b. Again, the value of UB is given by $\sum_{i \in N}(n+i-1) = \frac{(n+1)n}{2}$. However, when n is odd, the value of LB is given by

$$\frac{n+1}{2} + \sum_{i \in N,\, i \geq \frac{n-1}{2}} 2(n+1-i) = \frac{n+1+2(\frac{n-1}{2}+1)(\frac{n-1}{2})}{2} = \frac{(n+1)^2}{4},$$

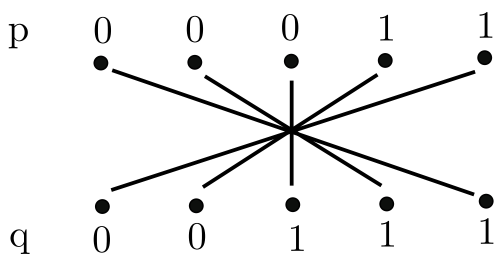
and therefore, when n is odd, we have that

$$\frac{UB}{LB} = \frac{2n(n+1)}{(n+1)^2} = \frac{2n}{n+1} \leq 2.$$

Thus, as $n \to \infty$, $\frac{UB}{LB} \to 2$.



(a) $n = 4$



(b) $n = 5$

Figure 8.3: Examples showing the constructed instances for $n = 4$ and $n = 5$.

We continue this analysis by looking at the range in which the true approximation ratio lies, as a function of Δ/n. In Figure 8.4, the line in blue shows the upper bound of 2 as stated by Theorem 8.4. The line in orange shows a lower bound of the approximation ratio, calculated as the actual approximation ratio for a specific instance, namely, a fully-crossed 0-1 instance with $n = 100$. The true approximation ratio is known to lie between these upper and lower bounds, where we already know that for $\Delta = 0$, the ratio 2 is tight, and for $\Delta = n$, (UB-SMSP) provides an optimal solution, i.e. the approximation guarantee becomes 1.

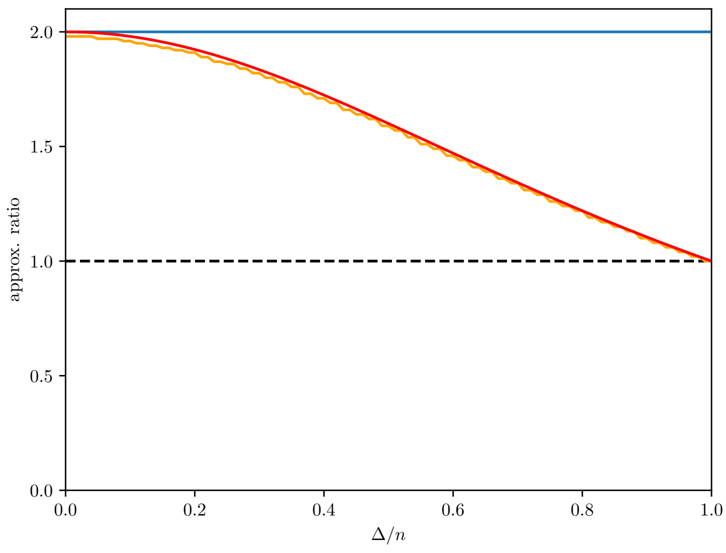


Figure 8.4: The upper bound on the approximation ratio of (UB-SMSP), as proved in Theorem 8.4, is shown in blue. A lower bound on the approximation ratio as a function of Δ/n, computed using a fully-crossed 0-1 instance with $n = 100$ is shown in orange. The lower bound as a function of Δ/n for fully-crossed 0-1 instances as $n \to \infty$ is shown in red.

Observe that an optimal solution to a fully-crossed 0-1 instance will always use the available $\lfloor \frac{n-\Delta}{2} \rfloor$ swaps on the outer-most jobs. See Figure 8.5 for an illustration of this. Knowing this, it becomes straightforward to compute the objective function of an optimal solution to such an instance. For example, in the case where $n - \Delta$ is even, the optimal objective value, v^*, is given by

$$\begin{aligned} v^* &= \underbrace{\frac{2\left(\frac{n-\Delta}{2}+1\right)\frac{n-\Delta}{2}}{2}}_{\text{outer } (n-\Delta)/2 \text{ swaps}} + \underbrace{\frac{\Big(\left(\frac{n-\Delta}{2}+\Delta\right)+\left(\frac{n-\Delta}{2}+1\right)\Big)\Delta}{2}}_{\text{inner } \Delta \text{ fixed jobs}} \\ &= \frac{n^2+\Delta^2+2n}{4}. \end{aligned}$$

Therefore, when $n - \Delta$ is even, the true approximation ratio can be computed as

$$\begin{aligned} \frac{UB}{v^*} &= \frac{2n(n+1)}{n^2+\Delta^2+2n} \\ &= \frac{2n^2+2n}{(1+\gamma^2)n^2+2n} \to \frac{2}{1+\gamma^2} \end{aligned}$$

as $n \to \infty$, where $\gamma = \Delta/n$. Note that a very similar analysis arrives at the same result for the case where $n - \Delta$ is odd. In Figure 8.4, we plot this limiting curve in red. Looking at the plot we see that for $\Delta = 0$, when the first and second solutions require no intersection, the upper and lower bounds converge as $n \to \infty$, confirming that the 2-approximation is tight. For $\Delta > 0$, it remains possible that still stronger guarantees can be found.

(UB-SMSP) is found by forcing the first and second-stage schedules to be the same. Clearly, this approach is overly conservative, and in practice it is beaten by solutions in which only a subset of jobs share a position across the two stages. The following result shows that the 2-approximation guarantee still holds for such solutions.

Lemma 8.6. *Function $f(M)$ is monotonically non-decreasing, i.e., $f(M) \leq f(M \cup \{i\})$ for any $i \in N \setminus M$.*

Proof. The optimisation problem (RecFix) has constraints for all $j \in M$. The

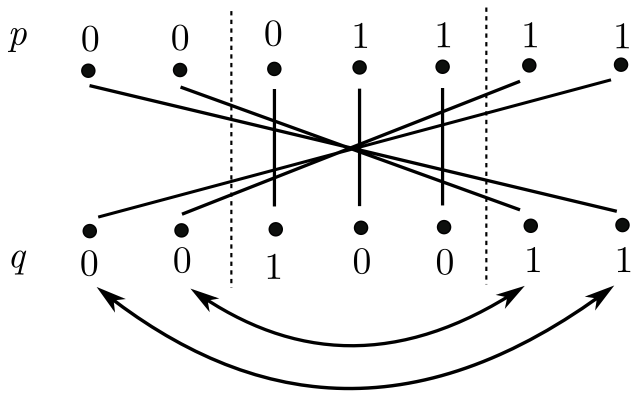


Figure 8.5: Optimal solution for a fully-crossed 0-1 instance where $n = 7$ and $\Delta = 3$. The $\lfloor \frac{n-\Delta}{2} \rfloor = 2$ available swaps are applied to the outer-most jobs. This structure is shared by the optimal solutions to all fully-crossed 0-1 instances.

problem of solving (RecFix) for M is therefore a relaxation of the problem of solving (RecFix) for $M \cup \{i\}$, and the claim follows. □

This means that any heuristic that gives a feasible pair of schedules based on Algorithm 8.1 is a 2-approximation as well.

Corollary 8.7. *For any $M \subseteq N$ with $|M| \geq \Delta$, the solution generated by Algorithm 8.1 gives a 2-approximation for (RecSMSP) in polynomial time.*

Corollary 8.7 suggests a natural greedy heuristic for solving (RecSMSP). This is outlined in Algorithm 8.2. The procedure begins by setting $M = \emptyset$ and iteratively adds to it the element $i \in N \setminus M$ with the smallest objective value $f(M \cup \{i\})$ to the set M, until $|X \cap Y| \geq \Delta$.

Due to Corollary 8.7, this greedy heuristic is ensured to be a 2-approximation. And due to Lemma 8.6, we also know that its objective value can be no worse than (UB-SMSP). Each iteration of the greedy heuristic takes $O(n^2 \log n)$ time and there are $O(\Delta)$ iterations, hence the heuristic runs in $O(n^3 \log n)$.

Algorithm 8.2 Greedy heurstic for (RecSMSP)

```
1: procedure GREEDY($\boldsymbol{p}, \boldsymbol{q}, \Delta$)
2:     initialise $M, X, Y = \emptyset$
3:     while $|X \cap Y| < \Delta$ do
4:         Set $v = \infty$
5:         for $j^{'} \in N \setminus M$ do
6:             $M^{'} \leftarrow M \cup \{j^{'}\}$
7:             $v^{'}$, $X$, $Y \leftarrow$ EVAL($\boldsymbol{p}, \boldsymbol{q}, M^{'}$)        ▷ Solve (RecFix) using EVAL
8:             if $v^{'} < v$ then
9:                 $v \leftarrow v^{'}$
10:                $j \leftarrow j^{'}$
11:            end if
12:        end for
13:        $M \leftarrow M \cup \{j\}$
14:    end while
15:    return $v$, $X$, $Y$
16: end procedure
```

8.4 Computational experiments

In this section, we compare results from solving the exact MIP formulation of (RecSMSP) with those of the 2-approximation provided by solving (UB-SMSP), as well from the greedy heuristic outlined at the end of the previous section.

Before presenting these results in detail, we outline the test instances used for these experiments and the computational hardware on which these experiments were performed. Section 8.4.1 investigates the performance of the MIP, focussing on its solution times and the gap to its linear relaxation. Following this, Section 8.4.2 looks at the 2-approximation and shows that in practice it considerably outperforms its theoretical worst-case performance. Finally, Section 8.4.3 presents results from the greedy heuristic.

For each value of $n \in \{10, 20, 50, 100\}$, 100 instances have been generated by randomly sampling p_i and q_i, $i \in N$, from the set $\{1, 2, \ldots, 100\}$. Experiments have been performed on these four instance sets for values of $\Delta \in \{0, 1, 2, \ldots, n\}$. These problem instances, in addition to the complete results data, can be downloaded from `https://github.com/boldm1/recoverable-robust-interval-SMSP`.

All the experiments have been run on 4 cores of a 2.30GHz Intel Xeon CPU, limited to 16GB RAM. The exact model has been solved using Gurobi 9.0.1, with a time limit of 20 minutes. The optimality gap tolerance (MIPGap) was changed to 1×10^{-6}; all other parameters were set to their default values.

8.4.1 MIP

We begin by considering the performance of the MIP formulation for (RecSMSP). Unsurprisingly, its performance depends on the size of n. However, more interestingly, it also depends on the number of free assignments across the first and second-stages, given by $n - \Delta$, and in particular whether this value is even or odd. We first present results that show the limitations of the MIP and demonstrate this dependence on the value of $n - \Delta$, before we then investigate the causes of this dependence by examining illustrative examples.

Note that since n is even for each of the instance sets we consider, even and odd values of Δ correspond to even and odd values of $n - \Delta$, respectively. Therefore, for ease of presentation, throughout this section we refer to the problem in terms of even and odd Δ, rather than in terms of even and odd $n - \Delta$.

For $n = 10$ and $n = 20$, the MIP was solved to optimality within the 20 minute time limit for all values of Δ for all instances. For $n = 50$, however, 5 instances for $\Delta = 47$ and 47 instances for $\Delta = 49$ were not solved to optimality within the time limit. And for $n = 100$, the number of instances that were not solved to optimality within the time limit for the values of Δ are shown in the Table 8.3. Note how it is only for certain odd values of Δ (and therefore odd values of $n - \Delta$) that the MIP cannot solve all instances.

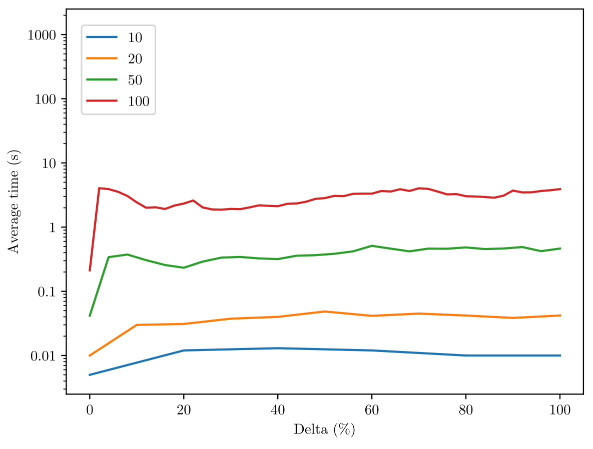
Δ	85	87	89	91	93	95	97	99
# non-opt.	2	3	9	17	43	57	83	100

Table 8.3: Number of instances with $n = 100$ for which the MIP could not be solved to optimality within 20 minutes, for different values of Δ. All instances were solved to optimality for the values of Δ not present in the table.

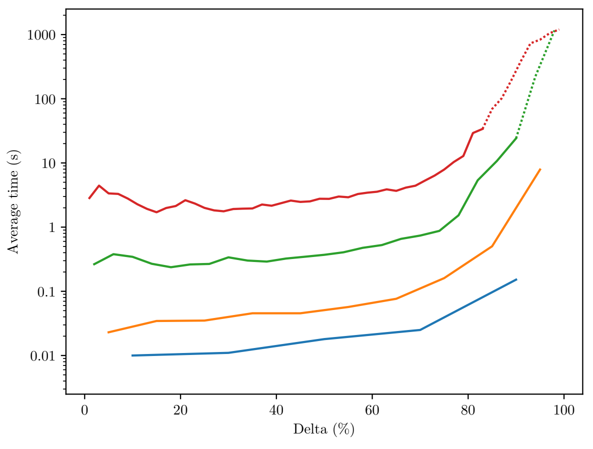
Figure 8.6 shows the average time to solve the MIP for even and odd values of Δ as a percentage of n, for $n \in \{10, 20, 50, 100\}$. A log-scale has been used for clarity. These plots show that solution times scale with n as expected. However, for odd values of Δ we see that solution times increase considerably for large Δ, whereas when Δ is even, it has little impact on solution times.

Clearly, the MIP cannot be easily solved for $n \geq 50$ for certain large values of Δ, and therefore the performance of the approximate solution approaches we propose are of genuine interest. Before we examine their performances over the following two sections, we investigate the marked difference in difficultly between instances with even and odd Δ.

We first consider the tightness of the MIP formulation by looking at the average gap between its solutions and the solutions of its linear relaxation (i.e. the average LP gap) in Figure 8.7. Observe that the average LP gaps for instances where Δ is even are orders of magnitudes smaller than for instances where Δ is odd; the



(a) Even Δ values.



(b) Odd Δ values.

Figure 8.6: Average time to solve the MIP for even and odd values of Δ as a percentage of n, for $n \in \{10, 20, 50, 100\}$. Line becomes dotted when not all instances were solved to optimality within the 1200s (20 minute) time limit.

largest single LP gap for an even-Δ instance and an odd-Δ instance for each of the instance sets are given in Table 8.4.

Figure 8.8 shows the percentage of instances for which the LP gap is non-zero, for even and odd values of Δ as a percentage of n, for $n \in \{10, 20, 50, 100\}$. Looking at this figure, we see that the proportion of instances with non-zero LP gaps is broadly similar for each of the four instance sets over the values of Δ. Therefore, to explain the results seen in Figure 8.7, it must be the case that the LP gaps become smaller as n increases. Figure 8.8 also shows that for larger values of Δ, the frequency of non-zero LP gaps increases dramatically for instances with odd values of Δ, but not for instances with even values of Δ.

In an attempt to shed light on this distinction, we consider the example with $n = 4$ jobs with binary processing times given in Figure 8.3a. Given Δ, we are free to allow up to $n - \Delta$ assignments to differ between the first and second-

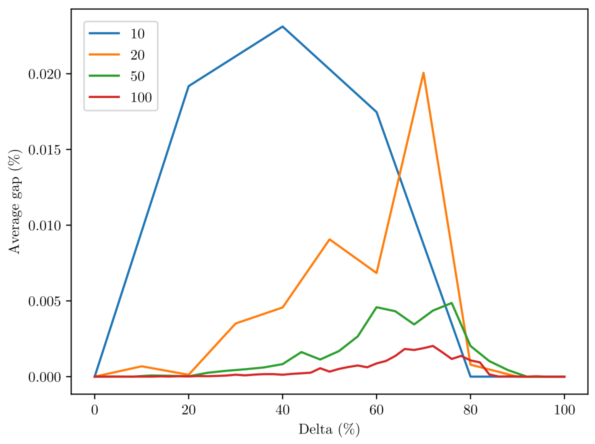
	LP gap (%)			
n	even-Δ		odd-Δ	
10	0.20	$(\Delta = 4)$	5.66	$(\Delta = 9)$
20	0.17	$(\Delta = 14)$	2.89	$(\Delta = 19)$
50	0.07	$(\Delta = 38)$	1.77	$(\Delta = 49)$
100	0.03	$(\Delta = 82)$	0.89	$(\Delta = 99)$

Table 8.4: Largest LP gap for a single even-Δ instance and odd-Δ instance for each of the instance sets.

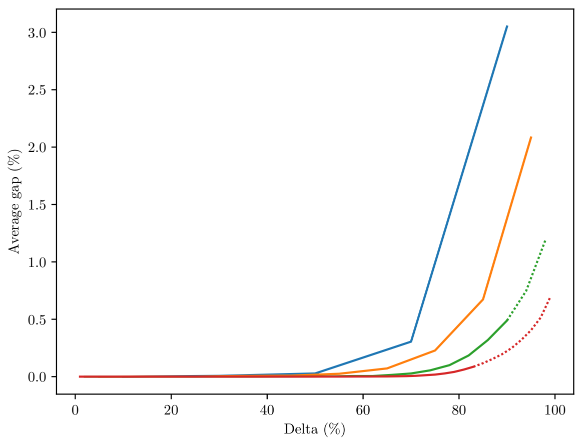
stage schedules. For clarity of presentation, in the following we assume that the first-stage schedule is a fixed horizontal assignment, and make changes to the assignments in the second-stage only.

Observe that for a given Δ, the best integral solution is found by swapping the positions of the jobs scheduled first and last, then swapping the positions of the jobs scheduled second and second-last, and so on, until no more swaps can be made without exceeding the $n - \Delta$ available free assignments. When $n - \Delta$ is even, the integral solution is able to change the assignments of exactly $n - \Delta$ jobs, and its linear relaxation finds the same solution and gains no advantage. When $n - \Delta$ is odd however, the integral solution is only able to change the assignments of $n - \Delta - 1$ jobs. In this case, the linear relaxation is able to make use of fractional assignments to beat the integral solution.

For example, when $\Delta = 1$, the MIP swaps the positions of only two jobs (Figure 8.9a). The linear relaxation gains an advantage over the integral solution since it can fractionally swap a further two jobs to make use of the last remaining free assignment (Figure 8.9b). In this instance the LP gap is given by $\frac{7-6.5}{6.5} \approx 7.7\%$.



(a) Even Δ values.



(b) Odd Δ values.

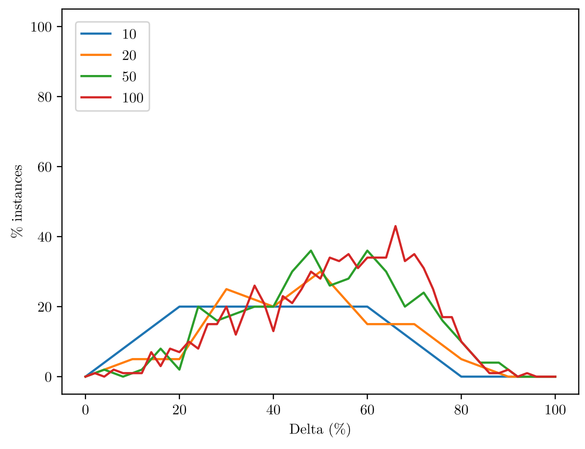
Figure 8.7: Average LP gap for even and odd values of Δ as a percentage of n, for $n \in \{10, 20, 50, 100\}$. Line becomes dotted when not all instances were solved to optimality within the 20 minute time limit.

Similarly, when $\Delta = 3$, the integral solution cannot feasibly swap the positions of any jobs (Figure 8.10a), whereas its linear relaxation is able to fractionally swap the positions of the first and last jobs (Figure 8.10b). The LP gap in this case is given by $\frac{10-8.5}{8.5} \approx$17.6%.

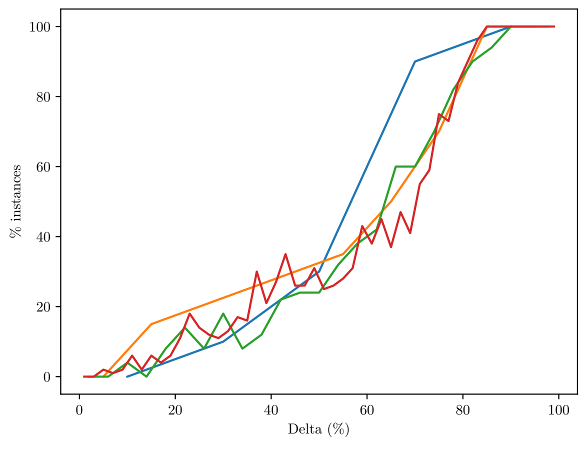
Experimental results seem to suggest that the maximum LP gap that can be achieved is 20%, which occurs for a fully-crossed instance with binary processing times when $n = 2$ and $\Delta = 1$ and when $n = 3$ and $\Delta = 2$.

8.4.2 UB-SMSP

As the results in the previous section demonstrate, large instances of (RecSMSP) cannot be solved within a reasonable time limit by the MIP. Therefore, accurate heuristic approximations are valuable for solving this problem. In this section, we look at the quality of the solutions to (UB-SMSP) in practice, to compare



(a) Even Δ values.



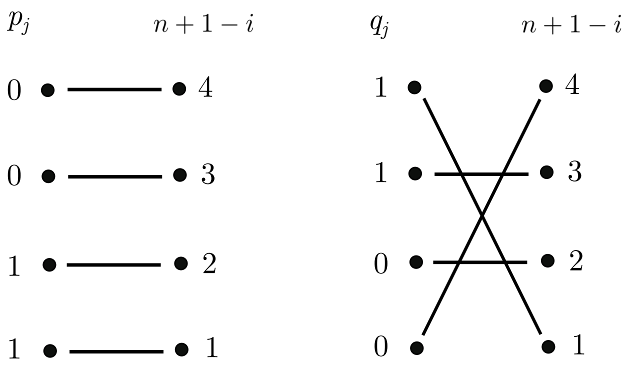
(b) Odd Δ values.

Figure 8.8: Percentage of instances for which the LP gap is non-zero, for even and odd values of Δ as a percentage of n, for $n \in \{10, 20, 50, 100\}$.

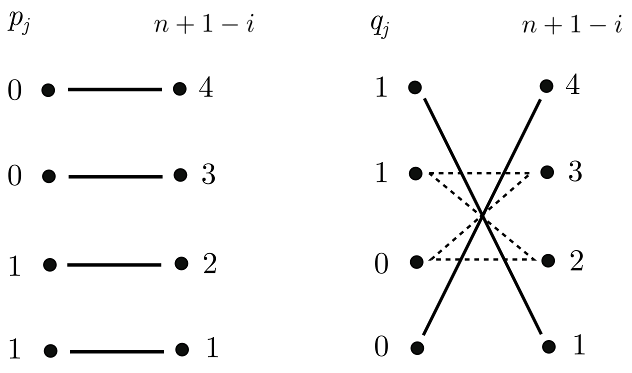
them against their theoretical worst-case performance ratio of 2. Recall that (UB-SMSP) can be solved trivially by ordering the jobs $j \in N$ by non-decreasing $p_j + q_j$ values. In our experiments, (UB-SMSP) was solved in less than 0.01 seconds for every instance.

Figure 8.11 shows the average relative gap between the solution to (UB-SMSP) and the solution to (RecSMSP) found by solving the MIP, plotted for values of Δ as a percentage of n, for $n \in \{10, 20, 50, 100\}$. A dotted line has been used to indicate the range of Δ values for which the MIP was not solved to optimality for all instances within the 20 minute time limit for the odd values of Δ within this range. The average gaps displayed for the odd values of Δ in this range are therefore estimates of the true average gap.

For all problem sizes the average gap decreases as Δ increases, reaching 0 when $\Delta = n$, where (RecSMSP) and (UB-SMSP) become equivalent. For $n = 10$, the largest gap for any single instance was 21.7%; for $n = 20$ the largest single gap was

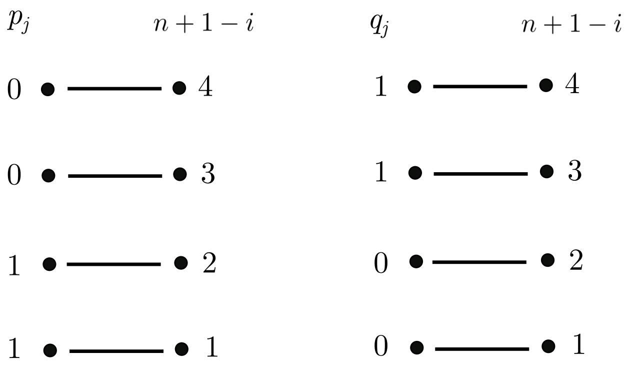


(a) Solution found by MIP for $\Delta = 1$ and $\Delta = 2$ and by linear relaxation for $\Delta = 2$. Objective value is $v = 3 + 4 = 7$.

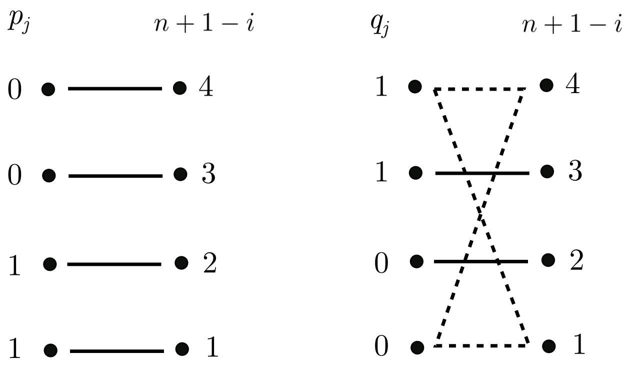


(b) Solution found by linear relaxation for $\Delta = 1$. Objective value is $v = 3 + 3.5 = 6.5$.

Figure 8.9: Solutions to instance shown in Figure 8.3a, found by MIP and its linear relaxation for $\Delta = 1$ and $\Delta = 2$.



(a) Solution found by MIP for $\Delta = 3$ and $\Delta = 4$ and by linear relaxation for $\Delta = 4$. Objective value is $v = 3 + 7 = 10$.



(b) Solution found by linear relaxation for $\Delta = 3$. Objective value is $v = 3 + 5.5 = 8.5$.

Figure 8.10: Solutions to instance shown in Figure 8.3a, found by MIP and its linear relaxation for $\Delta = 3$ and $\Delta = 4$.

18.2%, for $n = 50$ it was 23.0%, and for $n = 100$ it was 21.1%. For each value of n, this largest single gap was attained when $\Delta = 0$. Clearly, the worst-case gaps we see in practice are considerably smaller than the theoretical worst-case gap of 100%.

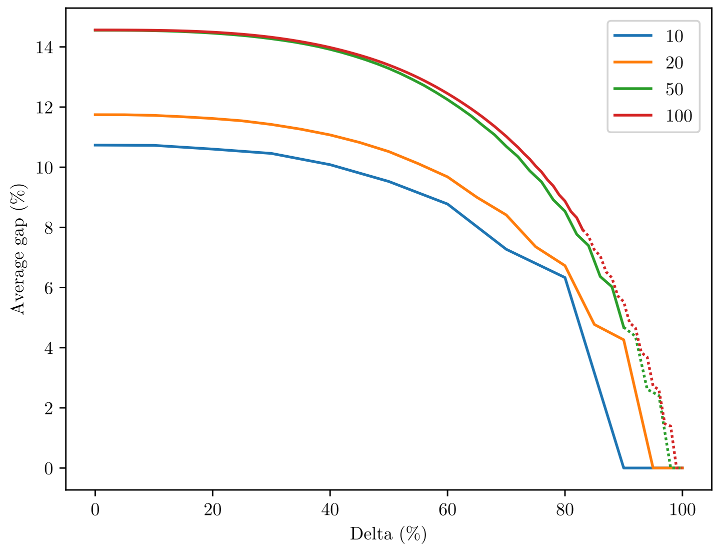


Figure 8.11: Average gap between solution to (UB-SMSP) and the solution to (RecSMSP) for values of Δ as a percentage of n, for $n \in \{10, 20, 50, 100\}$. Line becomes dotted for the range of Δ values for which not all instances were solved to optimality by the MIP within its 20 minute time limit for the odd values of Δ within this range.

8.4.3 Greedy heuristic

Finally, we examine the performance of greedy heuristic presented in Algorithm 8.2 for solving (RecSMSP). Recall that, as a result of Corollary 8.7, the objective value of solutions found by the greedy heuristic can be no worse than the solutions to (UB-SMSP), and as these results show, in practice they are considerably better than this.

We first look at the average run time of the greedy heuristic, shown in Figure 8.12 for values of Δ as a percentage of n, for each of the four instance sets. As expected, we see that the average run time grows with n and with Δ. The longest run time of any single instance was 4.72 seconds, which occurred for an instance with $n = 100$, $\Delta = 95$.

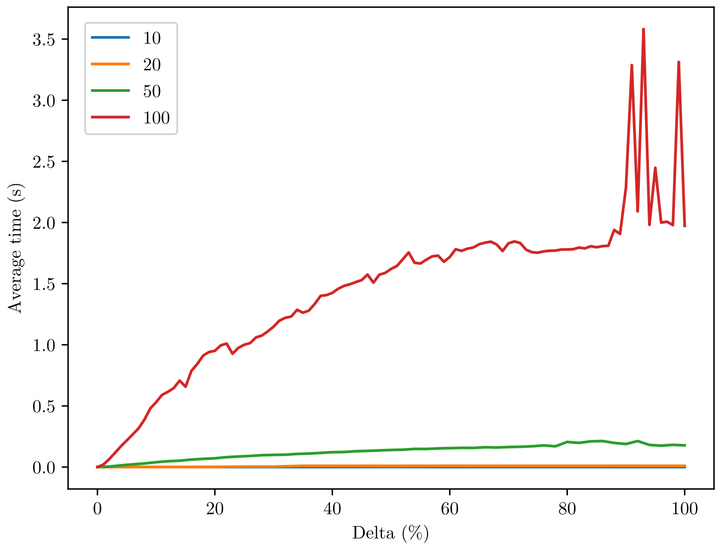


Figure 8.12: Average greedy heuristic run time for values of Δ as a percentage of n, for $n \in \{10, 20, 50, 100\}$.

Figure 8.13 shows the average relative gap between the solution found by the greedy heuristic and the solution to the MIP for values of Δ as a percentage of n, for $n \in \{10, 20, 50, 100\}$. A dotted line connects the data points at each value of Δ, whilst a solid line connects the data points corresponding to odd values of Δ only. The figure shows that the accuracy of the greedy heuristic varies depending on whether Δ is odd or even, demonstrated by the characteristic zig-zagging of the dotted line.

Even as the average gap between the greedy heuristic and the MIP increases with Δ, this gap is bounded above by the average gap between the solutions to (UB-SMSP) and the MIP, which sharply decreases with Δ (see Figure 8.11). The greedy heuristic finds solutions that are an order of magnitude closer to optimal than the solutions to (UB-SMSP). For $n = 10$ the maximum gap between the solution found by the greedy heuristic and the MIP solution for a single instance is 0.55%, for $n = 20$ it is 0.59%, for $n = 50$ it is 0.33%, and for $n = 100$ it is 0.18%. Hence, solutions found by the greedy heuristic are nearly optimal.

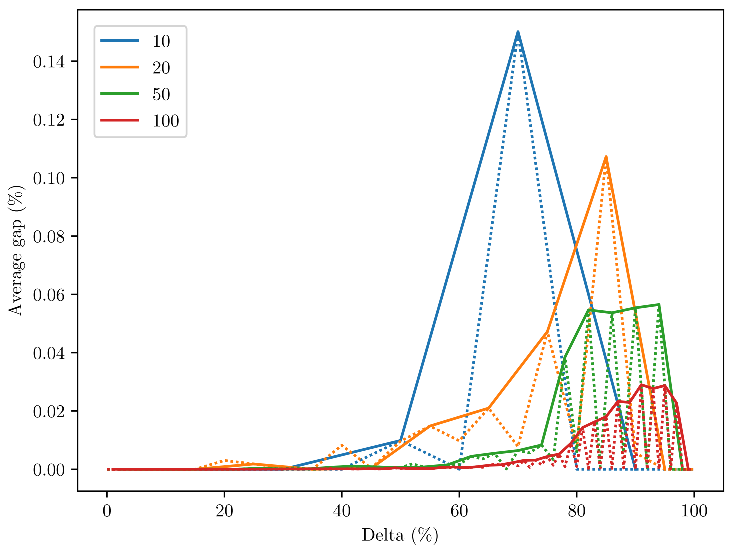


Figure 8.13: Average gap between greedy and MIP solution for values of Δ as a percentage of n, for $n \in \{10, 20, 50, 100\}$, plotted as a dotted line. Odd-Δ data points are connected with a solid line.

8.5 Conclusions

This chapter has presented a theoretical and computational investigation of the recoverable robust single machine scheduling problem under interval uncertainty with the objective of minimising the sum of completion times, (RecSMSP). Firstly, a number of positive complexity results have been produced for key subproblems. Specifically, we present a polynomial time sorting algorithm in Algorithm 8.1 which can optimally solve the case where a subset of jobs that must share a first and second-stage assignment is given. Using this result, we show that the special cases of (RecSMSP) with a constant value of Δ and with a constant number of possible processing times are solvable in polynomial time. We then introduce the problem (UB-SMSP) in which the first and second-stage assignments are forced to be identical, and prove that the solution to this problem provides a 2-approximation of (RecSMSP). This guarantee extends to the solutions found by a simple and fast greedy heuristic. A set of extensive computational experiments investigate the limitations of an exact mixed-integer programming approach to solving the problem, show that the solutions of (UB-SMSP) significantly outperform their worst-case guarantee in practice, and demonstrate the strength of the proposed greedy heuristic.

Note that the computational complexity of (RecSMSP) has yet to be characterised and remains unknown. In addition to investigating this, a promising direction for future research on this problem would be to consider the impact of alternative uncertainty sets on its complexity. Finally, another possibility for future research would be to look at this recoverable robust model for single machine scheduling problems with objective criteria different to the sum of completion times that we consider here.

Chapter 9

Conclusions

This thesis has focused on the development and analysis of new models and solution approaches for a range of different deterministic and uncertain scheduling problems. The main content in chapters 4-8 can be broadly grouped by the three categories of problem that they consider, relating to deterministic project scheduling, two-stage robust project scheduling, and two-stage robust single machine scheduling, respectively. In this concluding chapter, the contributions made in this thesis to each of these domains are reiterated and some of the most promising directions for future research that would build upon this work are highlighted.

We first summarise the contributions of Chapter 4, in which a new deterministic resource-constrained project scheduling model is proposed. This chapter, for the first time, combines generalised precedence constraints and flexible resource allocation into a single model, referred to as the generalised flexible resource-constrained project scheduling problem. The creation of this new model is directly motivated by the need to schedule the decommissioning of the Sellafield nuclear site in North West England. The complex inter-dependencies between the site facilities require the consideration of generalised precedence constraints, whilst the very long durations of some of the decommissioning activities prompts the inclusion of flexible resource allocation. As well as presenting a MILP formulation to solve this model, a genetic algorithm built upon a non-greedy flexible schedule generation scheme with an unscheduling step is also proposed. A series of computational experiments assess the contributions of the different components of the proposed genetic algorithm, and show that it performs competitively with the MILP on small test instances and outperforms it on larger instances. Chapter 4 concludes by modelling and solving a version of the Sellafield decommissioning scheduling problem, highlighting the relevance of the developed model and further demonstrating the strength of the proposed metaheuristic algorithm.

Given the severe complexity of this problem, the development of further improvements to the metaheuristic approaches proposed in Chapter 4 would be of significant value. Such improvements would likely employ a hybrid metaheuristic strategy that might, for example, include an additional local search heuristic, as has been repeatedly used to great effect for solving more classical project scheduling problems.

Regarding two-stage robust project scheduling, Chapters 5 and 6 develop and implement the first compact MILP formulations for the two-stage robust RCPSP and MRCPSP under uncertain activity durations. The specific model that is considered in these chapters assumes that activity durations lie in a budgeted uncertainty set and follows a two-stage decision process in which a decision maker must resolve resource conflicts subject to the problem uncertainty, but can determine activity start times after the uncertain activity durations become known. Chapter 5 applies this model to the RCPSP and compares the results of solving this model against the current state-of-the-art decomposition-based method proposed by Bruni et al. (2018). Chapter 6 extends the formulation to consider multiple activity processing modes and compares its performance against the Benders'-style approach of Balouka and Cohen (2021). In both cases, results show that the newly developed compact robust formulation can be implemented and solved straightforwardly using off-the-shelf optimisation software considerably faster than the previous state-of-the-art decomposition-based solution approaches, whilst significantly extending the range of instances that can be solved to optimality.

A number of possibilities to build on the research in Chapters 5 and 6 come to mind. Firstly, it would be of great interest to apply this approach in a real-world setting and demonstrate the practical importance of this two-stage robust scheduling approach. However, despite the dominance of the new compact formulations, they can only be solved for moderately-sized instances, limiting their application to real-world problems. As it stands, no heuristic approach of any kind has been proposed to solve the two-stage robust RCPSP or MRCPSP. Therefore, research into the development of heuristic approaches for solving these problems may prove very fruitful, and importantly would enable their solution on a larger and more realistic scale. Furthermore, the two-stage robust RCPSP and MRCPSP problems have only ever been considered in the context of continuous budgeted uncertainty. It would therefore be of significant interest to consider some alternative models of uncertainty, such as ellipsoidal, general polyhedral, or discrete scenario uncertainty. However, given that the reformulation derived in Chapter 5 relies on the linearity of the adversarial subproblem resulting from the budgeted uncertainty set, inevitably this would require the development of an alternative solution approach to the one proposed in this thesis.

Finally, Chapter 7 and 8 extend the recoverable robust optimisation framework to single machine scheduling for the first time. In the problems considered in both chapters, a first-stage schedule must be determined subject to uncertain job processing times, but following the realisation of the actual data, the schedule can be amended in some limited fashion. These two chapters differ in their choice of uncertainty set, as well as in the limitations placed on the recovery action.

Chapter 7 considers the case of general polyhedral uncertainty and allows a limited number of disjoint pairs of activities to be swapped to obtain a second-stage schedule. This problem is analysed in detail leading to a general result for matching problems with a specific cost structure, and three compact MILP formulations. These formulations are compared computationally for the case of continuous budgeted uncertainty and the superior strength of one of them is demonstrated. Regarding future research on this problem, the most pressing extension would be the relaxation of the recourse action to allow for the interchanging of three or more jobs. However, much of the analysis in this chapter relies on the specific choice of recourse action, and so an approach to the analysis of this problem is not immediately clear.

One such attempt is made in Chapter 8, however in this case the complexity of the problem is reduced by considering interval uncertainty. The more general recovery action considered in this chapter simply requires a certain number of jobs to share the same position in the first and second-stage schedule. Chapter 8 provides positive complexity results for some important special cases for this problem, and a 2-approximation is presented for the full problem. Computational experiments demonstrate the strong performance of a proposed polynomial-time greedy heuristic. However, it is important to highlight that the computational complexity of the full problem remains unknown. Hence an obvious avenue for future work would be the characterisation of the complexity of this problem, and others similar to it. Furthermore, both of the robust single machine scheduling problems that have been analysed in Chapters 7 and 8 consider the total sum of completion times objective function. In the context of single-stage robust optimisation, numerous other objective functions have received significant attention, so it would seem quite natural to extend the analysis of these problems to the two-stage recoverable robust setting. Such objective functions include weighted sum of completion times, number of late jobs and maximum tardiness. Simply put, the huge range of possibilities for combining alternative objective functions, recovery actions and uncertainty sets should serve to produce an abundance of interesting recoverable robust single machine scheduling problems, and two-stage robust scheduling problems more generally, for a long time come.

Appendix A

GFRCPSP Pre-processing Procedure

The following pre-processing procedure was applied to each instance before it was solved by any of the solution methods. This procedure makes a basic check regarding the feasibility of the instance with respect to the project precedence constraints, calculates the earliest and latest start and finish times of each activity $i \in N$, as well as an upper bound on the minimum project makespan, and records all the cycles in the project precedence network. The calculation of these parameters significantly reduces the number of variables required by the MIP, as well as reduces the number of iterations required by the FSGS in Algorithm 3.1.

The basis of this pre-processing procedure is the propagation of the individual precedence constraints across the project network to find the tightest constraint between each pair of activities. This information is contained in a distance matrix $D = (d_{kl}) \in \mathbb{R}^{|K|\times|K|}$, where $K = \{0, 1, \ldots, 2n, 2n+1\}$ is the set of start and finish events of the project activities, and where d_{kl} is the largest minimum time-lag between the events k and l. Matrix D is initialised to contain the information in the project network by setting $d_{kk} = 0$ for each $k \in K$, setting $d_{kl} = a_{kl}$ if there is a minimum time-lag of length a_{kl} from event k to event l, and setting $d_{kl} = -M$

otherwise, where M is some number larger than $T^{\max}$. Having initialised matrix D in this way, the Floyd-Warshall algorithm is applied to compute the tightest minimum time-lag between each pair of events. This runs in $O(n^3)$ time.

Having applied the Floyd-Warshall algorithm to matrix D, it can be immediately determined whether or not the project is feasible with respect to the project precedence constraints by looking at the diagonal elements of D. If any diagonal element is positive, the project contains a cycle of positive length and is therefore infeasible.

The earliest and latest start and finish times of each activity $i \in N$ can easily be obtained from matrix D since $ES_i = d_{0,S_i}$, $LS_i = -d_{S_i,0}$, $EF_i = d_{0,F_i}$, and $LF_i = -d_{F_i,0}$, where S_i and F_i are the indices of matrix D that represent the start and finish events of activity i, respectively.

Following the computation of the distance matrix D, an upper bound on the project makespan is calculated as:

$$T^{\max} = \sum_{i \in N} \max \Bigg\{ \overline{d}_i, \max_{(i,j,a_1) \in E_{SS}} a_1, \max_{(i,j,a_2) \in E_{SF}} \Big(a_2 - \underline{d}_j \Big), \max_{(i,j,a_3) \in E_{FS}} \Big(\overline{d}_i + a_3 \Big), \max_{(i,j,a_4) \in E_{FF}} \Big(\overline{d}_i + a_4 - \underline{d}_j \Big) \Bigg\}.$$

In the final step of the pre-processing procedure, the cycles in the project network are found and recorded using a depth-first search approach in $O(n + |E|)$ time, where $E = E_{SS} \cup E_{SF} \cup E_{FS} \cup E_{FF}$.

Appendix B

Supplementary Material for Chapter 5

B.1 Non-integrality of the adversarial sub-problem

Here, we show that the constraint matrix of model (5.13)-(5.22) is not totally unimodular, contrary to the claim made in Bruni et al. (2017). In the following, we define $\mathcal{E} := E \cup X$. The constraint matrix of (5.13)-(5.22) can be written in matrix notation as:

$$C = \begin{array}{c} \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{ccc} \alpha & w & \delta \\ \end{array}$$

$$C = \left(\begin{array}{ccc} A & 0 & 0 \\ 0 & I_{\mathcal{E}} & -B \\ -I_{\mathcal{E}} & I_{\mathcal{E}} & 0 \\ 0 & 0 & e_V^T \\ 0 & 0 & I_V \end{array}\right) \begin{array}{l} \text{Group 1 (5.14)-(5.16)} \\ \text{Group 2 (5.17)} \\ \text{Group 3 (5.18)} \\ \text{Group 4 (5.19)} \\ \text{Group 5 (5.20)} \end{array}$$

where A is a $|V| \times |\mathcal{E}|$ arc-node incidence matrix, B is a $|\mathcal{E}| \times |V|$ matrix where $B_{(i,j),i} = 1$ for each $(i,j) \in \mathcal{E}$, and 0 otherwise, I_V and $I_{\mathcal{E}}$ are identity matrices of dimension $|V|$ and $|\mathcal{E}|$ respectively, and e_V is a $|V| \times 1$ vector of 1's. The rows of this matrix have been grouped according to the constraints that they represent, and similarly, the columns have been grouped by the variables that they represent.

Ghouila-Houri (1962) showed that a matrix A is totally unimodular if and only if for every subset of rows R, there exists a partition of R into two disjoint subsets R_1 and R_2 such that

$$\sum_{i \in R_1} a_{ij} - \sum_{i \in R_2} a_{ij} \in \{-1, 0, 1\}, \quad \forall j = 1, \dots, n.$$

Therefore, finding a subset of rows of matrix C for which this condition cannot hold will prove that C is not totally unimodular.

Consider the constraint matrix of the example shown in Figure 5.3:

$$C = \begin{array}{cccc|cccc|cccc|l}
\multicolumn{4}{c}{\overbrace{\hspace{6em}}^{\alpha}} & \multicolumn{4}{c}{\overbrace{\hspace{6em}}^{w}} & \multicolumn{4}{c}{\overbrace{\hspace{6em}}^{\delta}} & \\
C1 & C2 & C3 & C4 & C5 & C6 & C7 & C8 & C9 & C10 & C11 & C12 & \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & R1 \; \Big\} \text{Group 1} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & R2 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & R3 \; \Big\} \text{Group 2} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\
-1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & R4 \\
0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & R5 \; \Big\} \text{Group 3} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \text{Group 4 + Group 5}
\end{array}$$

Take R to be the subset of rows consisting of the first row of Group 1, and first two rows of Groups 2 and 3. We will refer to these rows as $R1, \dots, R5$. To ensure that the sum of column $C9$ is in $\{-1, 0, 1\}$, $R2$ and $R3$ must be assigned opposite signs. $R4$ and $R5$ must have opposite signs to $R2$ and $R3$, respectively to ensure that the sum of columns $C5$ and $C6$ are in $\{-1, 0, 1\}$. Then, whatever the choice of sign for $R1$, the sum of column $C1$ and the sum of column $C2$ cannot both be in $\{-1, 0, 1\}$. Hence, there exists a subset of rows for which the Ghouila-Houri characterisation of total unimodularity does not hold, thus proving that matrix C is not totally unimodular, and that model (5.13)-(5.22) is not equivalent to its linear relaxation.

B.2 Example of the warm-start procedure

As an example, we apply the warm-start procedure from Section 5.4.2 to the project given in Figure 5.1. We set $T^{\max} = 20$ as an arbitrary upper bound on the minimum makespan of the corresponding deterministic project. The earliest-start-times ES_i, $i \in V$ are calculated via a forward-pass of the project precedence network, whilst the latest-finish-times LF_i, $i \in V$ are calculated relative to $T^{\max}$ via a backward-pass of the project precedence network. These values are shown in Figure B.1.

The LFT priority-rule heuristic orders the project activities according to increasing latest-finish-times to get $\sigma = (0, 2, 1, 4, 5, 3, 6, 7, 8)$, and then obtains a feasible schedule by scheduling these activities one-at-a-time in the order that they appear in σ using the serial schedule generation scheme (Kolisch, 1996b). This results in the following feasible schedule: $S_0 = 0$, $S_1 = 0$, $S_2 = 0$, $S_3 = 6$, $S_4 = 3$, $S_5 = 3$, $S_6 = 11$, $S_7 = 12$, $S_8 = 16$.

From this schedule, the set of feasible y-variables obtained is shown in Table B.1. Formulation (5.34)-(5.44) is solved with y-variables fixed to these values to obtain feasible values for the S-variables and f-variables. The resulting solution is the feasible warm-start solution we use.

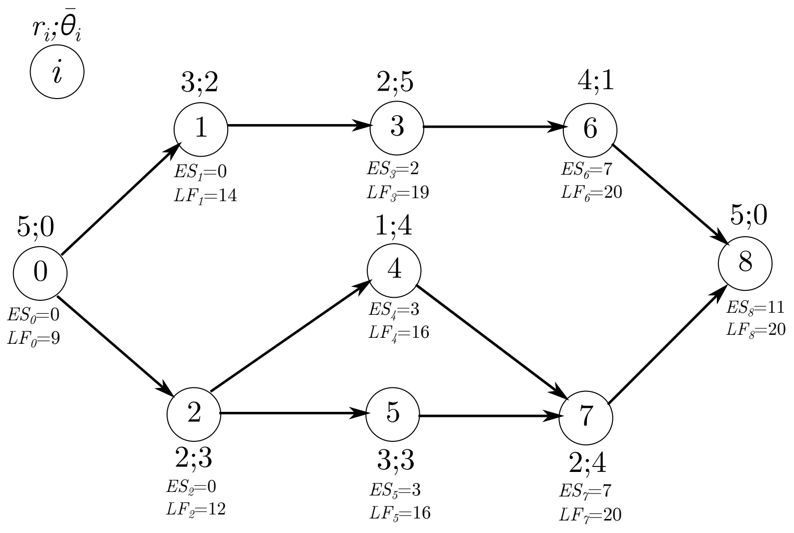


Figure B.1: Earliest start times and latest finish times for the deterministic project corresponding to the example project shown in Figure 5.1. Latest start times have been calculated relative to an arbitrary upper bound on the minimum project makespan given by $T^{\max} = 20$.

						j				
		0	1	2	3	4	5	6	7	8
	0	1	1	1	1	1	1	1	1	1
	1	0	0	0	1	1	1	1	1	1
	2	0	0	0	1	1	1	1	1	1
	3	0	0	0	0	0	0	1	1	1
i	4	0	0	0	0	0	0	1	1	1
	5	0	0	0	1	0	0	1	1	1
	6	0	0	0	0	0	0	0	1	1
	7	0	0	0	0	0	0	0	0	1
	8	0	0	0	0	0	0	0	0	1

Table B.1: The set of feasible y-variables corresponding to the solution found by the LFT priority-rule heuristic for the example project in Figure B.1. The ij-th element of this table gives the value of variable y_{ij}.

Appendix C

Supplementary Material for Chapter 7

C.1 Omitted proofs

C.1.1 Proof of Theorem 7.6

Let an instance I of problem (7.41)-(7.43) be given, and define

$$P = \{\boldsymbol{y} \in \mathbb{R}_{+}^{n \times n} : (7.36) - (7.39)\}.$$

To solve I, we construct a graph with a single node for each job $i \in \mathcal{N}$, where an edge between nodes i and j indicates that jobs i and j swap their positions from the first-stage schedule. Since edges correspond to unique swaps, the set of edges in this graph is given by $\mathcal{E} = \{(i,j) : i, j \in \mathcal{N},\ j > i\}$. The weight of an edge (i,j) in this graph is equal to the reduction in objective cost from making the corresponding swap, i.e. $a_i b_i + a_j b_j - a_i b_j - a_j b_i = (a_i - a_j)(b_i - b_j)$. We aim to choose up to Δ edges from this graph to maximise the reduction in objective cost. That is, given I, we construct an instance J of the following cardinality-constrained matching problem:

$$
\begin{aligned}
\min\ & \sum_{i\in\mathcal{N}} a_i b_i - \sum_{(i,j)\in\mathcal{E}} (a_i - a_j)(b_i - b_j) z_{ij} && && \text{(C.1)}\\
\text{s.t. } & \sum_{(i,j)\in\mathcal{E}} z_{ij} + \sum_{(j,i)\in\mathcal{E}} z_{ij} \le 1 && \forall i \in \mathcal{N} && \text{(C.2)}\\
& \sum_{(i,j)\in\mathcal{E}} z_{ij} \le \Delta && && \text{(C.3)}\\
& z_{ij} \ge 0 && \forall (i,j) \in \mathcal{E}. && \text{(C.4)}
\end{aligned}
$$

As stated in Corollary 7.4, this problem has an integral optimal solution. Hence, letting $P' = \{\boldsymbol{z} \in \mathbb{R}_+^{|\mathcal{E}|} : (\text{C.2}) - (\text{C.4})\}$, we construct mappings $\phi : P \to P'$ and $\phi^{-1} : P' \to P$ which preserve objective value and integrality, showing problems I and J are indeed equivalent. To this end, we define $\phi(\boldsymbol{y}) = \boldsymbol{z}$ by $z_{ij} = y_{ij}$ for all $(i,j) \in \mathcal{E}$. Observe that

$$
\begin{aligned}
obj_I(\boldsymbol{y}) &= \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}} a_i b_j y_{ij}\\
&= \sum_{i\in\mathcal{N}} a_i b_i y_{ii} + \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}:j\neq i} a_i b_j y_{ij}\\
&= \sum_{i\in\mathcal{N}} a_i b_i y_{ii} + \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}:j> i} (a_i b_j + a_j b_i) y_{ij}\\
&= \sum_{i\in\mathcal{N}} a_i b_i \left(1 - \sum_{j\in\mathcal{N}:j\neq i} y_{ij}\right) + \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}:j> i} (a_i b_j + a_j b_i) y_{ij}\\
&= \sum_{i\in\mathcal{N}} a_i b_i - \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}:j\neq i} a_i b_i y_{ij} + \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}:j> i} (a_i b_j + a_j b_i) y_{ij}\\
&= \sum_{i\in\mathcal{N}} a_i b_i - \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}:j> i} (a_i b_i + a_j b_j) y_{ij} + \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}:j> i} (a_i b_j + a_j b_i) y_{ij}\\
&= \sum_{i\in\mathcal{N}} a_i b_i - \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}:j> i} (a_i b_i + a_j b_j - a_i b_j - a_j b_i) y_{ij}
\end{aligned}
$$

$$= \sum_{i \in \mathcal{N}} a_i b_i - \sum_{(i,j) \in \mathcal{E}} (a_i - a_j)(b_i - b_j) z_{ij}$$

$$= obj_J(\boldsymbol{z}).$$

Conversely, we define $\phi^{-1}(\boldsymbol{z}) = \boldsymbol{y}$ with $y_{ij} = y_{ji} = z_{ij}$ for each $(i,j) \in \mathcal{E}$, and $y_{ii} = (1 - \sum_{j \in \mathcal{N}: j > i} z_{ij})(1 - \sum_{j \in \mathcal{N}: j < i} z_{ji})$ for each $i \in \mathcal{N}$. Then

$$\begin{aligned}
obj_J(\boldsymbol{z}) &= \sum_{i \in \mathcal{N}} a_i b_i - \sum_{(i,j) \in \mathcal{E}} (a_i - a_j)(b_i - b_j) z_{ij} \\
&= \sum_{i \in \mathcal{N}} a_i b_i - \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}: j > i} (a_i b_i + a_j b_j) z_{ij} + \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}: j > i} (a_i b_j + a_j b_i) z_{ij} \\
&= \sum_{i \in \mathcal{N}} a_i b_i - \sum_{i \in \mathcal{N}} a_i b_i \sum_{j \in \mathcal{N}: j > i} z_{ij} - \sum_{i \in \mathcal{N}} a_i b_i \sum_{j \in \mathcal{N}: j < i} z_{ji} \\
&\qquad + \sum_{i \in \mathcal{N}} a_i b_i \underbrace{\sum_{j \in \mathcal{N}: j > i} z_{ij} \sum_{j \in \mathcal{N}: j < i} z_{ji}}_{=0} + \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}: j > i} (a_i b_j + a_j b_i) z_{ij} \\
&= \sum_{i \in \mathcal{N}} a_i b_i \left(1 - \sum_{j \in \mathcal{N}: j > i} z_{ij}\right)\left(1 - \sum_{j \in \mathcal{N}: j < i} z_{ji}\right) + \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}: j > i} (a_i b_j + a_j b_i) z_{ij} \\
&= \sum_{i \in \mathcal{N}} a_i b_i y_{ii} + \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}: j > i} a_i b_j y_{ij} + \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}: j > i} a_j b_i y_{ji} \\
&= \sum_{i \in \mathcal{N}} a_i b_i y_{ii} + \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}: j \neq i} a_i b_j y_{ij} \\
&= \sum_{i \in \mathcal{N}} a_i b_j y_{ij} \\
&= obj_I(\boldsymbol{y}).
\end{aligned}$$

Therefore $\min_{\boldsymbol{y} \in P} obj_I(\boldsymbol{y}) = \min_{\boldsymbol{z} \in P'} obj_J(\boldsymbol{z})$. Since problem (7.24)-(7.27) has an optimal integral solution, there must also be an optimal integral solution to I via the mapping ϕ^{-1}, proving the claim. $\square$

C.1.2 Proof of Theorem 7.8

To prove this, it will suffice to show that $P(F_m) \subsetneq P(F_a)$, where $P(F_m)$ and $P(F_a)$ denote the sets of feasible solutions to the linear relaxations of the non-linear matching-based and assignment-based formulations respectively, i.e. that for each $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{q}) \in P(F_m)$, there exists a $\boldsymbol{y}$ such that $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{q}) \in P(F_a)$. Note that the same vector $\boldsymbol{q}$ is used in both formulations, which results in the objective values being the same. To this end, let $\boldsymbol{z}$ be part of a feasible solution to the matching formulation, and define the transformation $\phi(\boldsymbol{z}) = \boldsymbol{y}$ by $y_{ij} = y_{ji} = z_{ij}$ for $(i, j) \in \mathcal{E} = \{(i, j) \in \mathcal{N} \times \mathcal{N} : i < j\}$ and $y_{ii} = 1 - \sum_{j \in \mathcal{N}: j>i} z_{ij} - \sum_{j \in \mathcal{N}: j<i} z_{ji}$ for $i \in \mathcal{N}$.

Firstly, observe that the assignment constraints (7.45) and (7.46) are satisfied by this definition of $\boldsymbol{y}$. For each $j \in \mathcal{N}$ we have that

$$\begin{aligned} \sum_{i \in \mathcal{N}} y_{ij} &= \sum_{i \in \mathcal{N}: i<j} y_{ij} + \sum_{i \in \mathcal{N}: i>j} y_{ji} + y_{ii} \\ &= \sum_{i \in \mathcal{N}: i<j} z_{ij} + \sum_{i \in \mathcal{N}: i>j} z_{ji} + 1 - \sum_{i \in \mathcal{N}: i<j} z_{ij} - \sum_{i \in \mathcal{N}: i>j} z_{ji} = 1. \end{aligned}$$

The same can be shown for the ‘incoming’ assignment constraints for each $i \in \mathcal{N}$.

Now observe that constraint (7.31) states that

$$\sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}: j>i} z_{ij} \leq \Delta,$$

or equivalently

$$\sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}: j>i} y_{ij} \leq \Delta.$$

Since, $y_{ij} = y_{ji}$ for all $i,\, j \in \mathcal{N}$, we have that

$$\sum_{i\in\mathcal{N}} \Bigg(\sum_{j\in\mathcal{N}:j>i} y_{ij} + \sum_{j\in\mathcal{N}:j<i} y_{ij} \Bigg) \leq 2\Delta,$$

and therefore as a consequence of the assignment constraints, which imply

$$\sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}} y_{ij} = \sum_{i\in\mathcal{N}} \Bigg(\sum_{j\in\mathcal{N}:j>i} y_{ij} + \sum_{j\in\mathcal{N}:j<i} y_{ij} + y_{ii} \Bigg) = n,$$

we have that

$$\sum_{i\in\mathcal{N}} y_{ii} = n - \sum_{i\in\mathcal{N}} \Bigg(\sum_{j\in\mathcal{N}:j>i} y_{ij} + \sum_{j\in\mathcal{N}:j<i} y_{ij} \Bigg) \geq n - 2\Delta.$$

This is exactly constraint (7.47) from the assignment-based formulation.

Finally, consider constraints (7.29):

$$\begin{aligned} &\sum_{m\in\mathcal{M}} a_{mi} q_m + \sum_{j\in\mathcal{N}:j>i} \Bigg(\sum_{\ell\in\mathcal{N}} \ell \cdot x_{j\ell} - \sum_{\ell\in\mathcal{N}} \ell \cdot x_{i\ell} \Bigg) z_{ij} \\ &- \sum_{j\in\mathcal{N}:j<i} \Bigg(\sum_{\ell\in\mathcal{N}} \ell \cdot x_{i\ell} - \sum_{\ell\in\mathcal{N}} \ell \cdot x_{j\ell} \Bigg) z_{ji} \geq (n + 1 - \sum_{\ell\in\mathcal{N}} \ell \cdot x_{i\ell}) \quad \forall i \in \mathcal{N}. \end{aligned}$$

These can be rewritten as

$$\begin{aligned} \sum_{m\in\mathcal{M}} a_{mi} q_m &\geq (n + 1 - \sum_{\ell\in\mathcal{N}} \ell \cdot x_{i\ell}) + \sum_{j\in\mathcal{N}:j<i} z_{ji} \sum_{\ell\in\mathcal{N}} \ell \cdot x_{i\ell} + \sum_{j\in\mathcal{N}:j>i} z_{ij} \sum_{\ell\in\mathcal{N}} \ell \cdot x_{i\ell} \\ &\qquad - \sum_{j\in\mathcal{N}:j<i} z_{ji} \sum_{\ell\in\mathcal{N}} \ell \cdot x_{j\ell} - \sum_{j\in\mathcal{N}:j>i} z_{ij} \sum_{\ell\in\mathcal{N}} \ell \cdot x_{j\ell} \\ &= (n + 1 - \sum_{\ell\in\mathcal{N}} \ell \cdot x_{i\ell}) + \sum_{j\in\mathcal{N}:j<i} y_{ij} \sum_{\ell\in\mathcal{N}} \ell \cdot x_{i\ell} + \sum_{j\in\mathcal{N}:j>i} y_{ij} \sum_{\ell\in\mathcal{N}} \ell \cdot x_{i\ell} \\ &\qquad - \sum_{j\in\mathcal{N}:j<i} y_{ij} \sum_{\ell\in\mathcal{N}} \ell \cdot x_{j\ell} - \sum_{j\in\mathcal{N}:j>i} y_{ij} \sum_{\ell\in\mathcal{N}} \ell \cdot x_{j\ell} \end{aligned}$$

$$\begin{aligned}
&= (n+1-\sum_{\ell\in\mathcal{N}}\ell\cdot x_{i\ell}) + \sum_{j\in\mathcal{N}:j\neq i} y_{ij}\sum_{\ell\in\mathcal{N}}\ell\cdot x_{i\ell} - \sum_{j\in\mathcal{N}:j\neq i} y_{ij}\sum_{\ell\in\mathcal{N}}\ell\cdot x_{j\ell}\\
&= n+1-\Big(1-\sum_{j\in\mathcal{N}:j\neq i} y_{ij}\Big)\sum_{\ell\in\mathcal{N}}\ell\cdot x_{i\ell} - \sum_{j\in\mathcal{N}:j\neq i} y_{ij}\sum_{\ell\in\mathcal{N}}\ell\cdot x_{j\ell}\\
&= n+1-y_{ii}\sum_{\ell\in\mathcal{N}}\ell\cdot x_{i\ell} - \sum_{j\in\mathcal{N}:j\neq i} y_{ij}\sum_{\ell\in\mathcal{N}}\ell\cdot x_{j\ell}\\
&= n+1-\sum_{j\in\mathcal{N}} y_{ij}\sum_{\ell\in\mathcal{N}}\ell\cdot x_{j\ell}\\
&= (n+1)\sum_{j\in\mathcal{N}} y_{ij} - \sum_{j\in\mathcal{N}} y_{ij}\sum_{\ell\in\mathcal{N}}\ell\cdot x_{j\ell}\\
&= \sum_{j\in\mathcal{N}}\big(n+1-\sum_{\ell\in\mathcal{N}}\ell\cdot x_{j\ell}\big)y_{ij},
\end{aligned}$$

for each $i\in\mathcal{N}$, which are constraints (7.49) from the assignment formulation.

Hence, every feasible solution to the non-linear matching-based formulation has a corresponding feasible solution for the non-linear assignment-based formulation, i.e. $P(F_m)\subseteq P(F_a)$. The examples used in the proof of Theorem 7.7 show that there exist instances for which the LP bound of the matching formulation is strictly larger than that of the assignment formulation, demonstrating that $P(F_m)\subset P(F_a)$. □

C.2 Complete formulations

C.2.1 General model

The complete linear compact formulation for the general recoverable robust model presented in Section 7.3 is as follows:

$$\min \sum_{m\in\mathcal{M}} b_m q_m \tag{C.5}$$

$$
\begin{aligned}
\text{s.t.}\quad & \sum_{k\in\mathcal{K}} \mu_k = 1 && && \text{(C.6)}\\
& \sum_{m\in\mathcal{M}} a_{mi} q_m \geq \sum_{k\in\mathcal{K}} \left(\sum_{j\in\mathcal{N}} (n+1-j) \sum_{i'\in\mathcal{N}} h^k_{ii'j} \right) && \forall i \in \mathcal{N} && \text{(C.7)}\\
& \sum_{i'\in\mathcal{N}} z^k_{ii'} = 1 && \forall i \in \mathcal{N},\, k \in \mathcal{K} && \text{(C.8)}\\
& \sum_{i\in\mathcal{N}} z^k_{ii'} = 1 && \forall i' \in \mathcal{N},\, k \in \mathcal{K} && \text{(C.9)}\\
& z^k_{ii'} = z^k_{i'i} && \forall i, i' \in \mathcal{N},\, k \in \mathcal{K} && \text{(C.10)}\\
& \sum_{i\in\mathcal{N}} z^k_{ii} \geq n - 2\Delta && \forall k \in \mathcal{K} && \text{(C.11)}\\
& \sum_{j\in\mathcal{N}} x_{ij} = 1 && \forall i \in \mathcal{N} && \text{(C.12)}\\
& \sum_{i\in\mathcal{N}} x_{ij} = 1 && \forall j \in \mathcal{N} && \text{(C.13)}\\
& w^k_{ii'j} \leq z^k_{ii'} && \forall i, i', j \in \mathcal{N},\, k \in \mathcal{K} && \text{(C.14)}\\
& w^k_{ii'j} \leq x_{i'j} && \forall i, i', j \in \mathcal{N},\, k \in \mathcal{K} && \text{(C.15)}\\
& w^k_{ii'j} \geq z^k_{ii'} + x_{i'j} - 1 && \forall i, i', j \in \mathcal{N},\, k \in \mathcal{K} && \text{(C.16)}\\
& h^k_{ii'j} \leq w^k_{ii'j} && \forall i, i', j \in \mathcal{N},\, k \in \mathcal{K} && \text{(C.17)}\\
& h^k_{ii'j} \leq \mu_k && \forall i, i', j \in \mathcal{N},\, k \in \mathcal{K} && \text{(C.18)}\\
& h^k_{ii'j} \geq \mu_k + w^k_{ii'j} - 1 && \forall i, i', j \in \mathcal{N},\, k \in \mathcal{K} && \text{(C.19)}\\
& w^k_{ii'j} \in \{0,1\} && \forall i, i', j \in \mathcal{N},\, k \in \mathcal{K} && \text{(C.20)}\\
& h^k_{ii'j} \geq 0 && \forall i, i', j \in \mathcal{N},\, k \in \mathcal{K} && \text{(C.21)}\\
& \mu_k \geq 0 && \forall k \in \mathcal{K} && \text{(C.22)}\\
& q_m \geq 0 && \forall m \in \mathcal{M} && \text{(C.23)}\\
& z^k_{ii'} \in \{0,1\} && \forall i,\, i' \in \mathcal{N},\, k \in \mathcal{K} && \text{(C.24)}\\
& x_{ij} \in \{0,1\} && \forall i,\, j \in \mathcal{N}. && \text{(C.25)}
\end{aligned}
$$

C.2.2 Matching-based model

The fully-linearised formulation of the matching-based model presented in Section 7.4.1 is as follows:

$$\min_{\boldsymbol{x},\boldsymbol{z},\boldsymbol{u},\boldsymbol{v},\boldsymbol{q}} \sum_{m\in\mathcal{M}} b_m q_m \tag{C.26}$$

$$\text{s.t.} \sum_{(i,j)\in\mathcal{E}} z_{ij} + \sum_{(j,i)\in\mathcal{E}} z_{ji} \le 1 \qquad \forall i\in\mathcal{N} \tag{C.27}$$

$$\sum_{(i,j)\in\mathcal{E}} z_{ij} \le \Delta \tag{C.28}$$

$$\begin{aligned}\sum_{m\in\mathcal{M}} a_{mi} q_m &+ \sum_{(i,j)\in\mathcal{E}} \left(\sum_{\ell\in\mathcal{N}} \ell\cdot v_{ij\ell} - \sum_{\ell\in\mathcal{N}} \ell\cdot u_{ij\ell}\right) \\ &- \sum_{(j,i)\in\mathcal{E}} \left(\sum_{\ell\in\mathcal{N}} \ell\cdot v_{ji\ell} - \sum_{\ell\in\mathcal{N}} \ell\cdot u_{ji\ell}\right) \\ &\ge (n+1-\sum_{\ell\in\mathcal{N}} \ell\cdot x_{i\ell}) \qquad \forall i\in\mathcal{N}\end{aligned} \tag{C.29}$$

$$\sum_{i\in\mathcal{N}} x_{i\ell} = 1 \qquad \forall \ell\in\mathcal{N} \tag{C.30}$$

$$\sum_{\ell\in\mathcal{N}} x_{i\ell} = 1 \qquad \forall i\in\mathcal{N} \tag{C.31}$$

$$u_{ij\ell} \le x_{i\ell} \qquad \forall (i,j)\in\mathcal{E},\ \ell\in\mathcal{N} \tag{C.32}$$

$$u_{ij\ell} \le z_{ij} \qquad \forall (i,j)\in\mathcal{E},\ \ell\in\mathcal{N} \tag{C.33}$$

$$u_{ij\ell} \ge z_{ij} + x_{i\ell} - 1 \qquad \forall (i,j)\in\mathcal{E},\ \ell\in\mathcal{N} \tag{C.34}$$

$$v_{ij\ell} \le x_{j\ell} \qquad \forall (i,j)\in\mathcal{E},\ \ell\in\mathcal{N} \tag{C.35}$$

$$v_{ij\ell} \le z_{ij} \qquad \forall (i,j)\in\mathcal{E},\ \ell\in\mathcal{N} \tag{C.36}$$

$$v_{ij\ell} \ge z_{ij} + x_{j\ell} - 1 \qquad \forall (i,j)\in\mathcal{E},\ \ell\in\mathcal{N} \tag{C.37}$$

$$u_{ij\ell} \ge 0 \qquad \forall (i,j)\in\mathcal{E},\ \ell\in\mathcal{N} \tag{C.38}$$

$$v_{ij\ell} \ge 0 \qquad \forall (i,j)\in\mathcal{E},\ \ell\in\mathcal{N} \tag{C.39}$$

$$q_m \geq 0 \qquad \forall m \in \mathcal{M} \tag{C.40}$$

$$z_{ij} \geq 0 \qquad \forall (i,j) \in \mathcal{E} \tag{C.41}$$

$$x_{i\ell} \in \{0,1\} \qquad \forall i, \ell \in \mathcal{N}. \tag{C.42}$$

C.2.3 Assignment-based model

The fully-linearised formulation of the assignment-based model derived in Section 7.4.2 is as follows:

$$\min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{w},\boldsymbol{q}} \sum_{m\in\mathcal{M}} b_m q_m \tag{C.43}$$

$$\text{s.t.} \sum_{i\in\mathcal{N}} y_{ij} = 1 \qquad \forall j \in \mathcal{N} \tag{C.44}$$

$$\sum_{j\in\mathcal{N}} y_{ij} = 1 \qquad \forall i \in \mathcal{N} \tag{C.45}$$

$$\sum_{i\in\mathcal{N}} y_{ii} \geq n - 2\Delta \tag{C.46}$$

$$y_{ij} = y_{ji} \qquad \forall i, j \in \mathcal{N} \tag{C.47}$$

$$\sum_{m\in\mathcal{M}} a_{mi} q_m \geq \sum_{j\in\mathcal{N}} \Big((n+1) y_{ij} - \sum_{\ell\in\mathcal{N}} \ell \cdot w_{ij\ell} \Big) \qquad \forall i \in \mathcal{N} \tag{C.48}$$

$$\sum_{i\in\mathcal{N}} x_{i\ell} = 1 \qquad \forall \ell \in \mathcal{N} \tag{C.49}$$

$$\sum_{\ell\in\mathcal{N}} x_{i\ell} = 1 \qquad \forall i \in \mathcal{N} \tag{C.50}$$

$$w_{ij\ell} \leq x_{j\ell} \qquad \forall i,\, j,\, \ell \in \mathcal{N} \tag{C.51}$$

$$w_{ij\ell} \leq y_{ij} \qquad \forall i,\, j,\, \ell \in \mathcal{N} \tag{C.52}$$

$$w_{ij\ell} \geq x_{j\ell} + y_{ij} - 1 \qquad \forall i,\, j,\, \ell \in \mathcal{N} \tag{C.53}$$

$$w_{ij\ell} \geq 0 \qquad \forall i,\, j,\, \ell \in \mathcal{N} \tag{C.54}$$

$$q_m \geq 0 \qquad \forall m \in \mathcal{M} \tag{C.55}$$

$$y_{ij} \geq 0 \qquad \forall i,\, j \in \mathcal{N} \tag{C.56}$$

$$x_{i\ell} \in \{0, 1\} \qquad \forall i,\, \ell \in \mathcal{N}. \tag{C.57}$$

Bibliography

Aissi, H., Aloulou, M. A., and Kovalyov, M. Y. (2011). Minimizing the number of late jobs on a single machine under due date uncertainty. *Journal of Scheduling*, 14(4):351–360.

Aissi, H., Bazgan, C., and Vanderpooten, D. (2009). Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European journal of operational research*, 197(2):427–438.

Alcaraz, J. and Maroto, C. (2001). A robust genetic algorithm for resource allocation in project scheduling. *Annals of operations Research*, 102(1):83–109.

Aloulou, M. A. and Della Croce, F. (2008). Complexity of single machine scheduling problems under scenario-based uncertainty. *Operations Research Letters*, 36(3):338–342.

Alumur, S. A., Nickel, S., and Saldanha-da Gama, F. (2012). Hub location under uncertainty. *Transportation Research Part B: Methodological*, 46(4):529–543.

Alvarez-Valdès, R. and Tamarit, J. M. (1993). The project scheduling polyhedron: Dimension, facets and lifting theorems. *European Journal of Operational Research*, 67(2):204–220.

Artigues, C., Demassey, S., and Neron, E., editors (2008). *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications.* ISTE/Wiley.

Artigues, C., Demassey, S., and Neron, E. (2013a). *Resource-constrained project scheduling: models, algorithms, extensions and applications*. John Wiley & Sons.

Artigues, C., Leus, R., and Nobibon, F. T. (2013b). Robust optimization for resource-constrained project scheduling with uncertain activity durations. *Flexible Services and Manufacturing Journal*, 25(1-2):175–205.

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

Averbakh, I. and Lebedev, V. (2004). Interval data minmax regret network optimization problems. *Discrete Applied Mathematics*, 138(3):289–301.

Balinski, M. L. (1965). Integer programming: Methods, uses, computations. *Management Science*, 12(3):253–313.

Ballestín, F., Barrios, A., and Valls, V. (2011). An evolutionary algorithm for the resource-constrained project scheduling problem with minimum and maximum time lags. *Journal of Scheduling*, 14(4):391–406.

Balouka, N. and Cohen, I. (2021). A robust optimization approach for the multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, 291(2):457–470.

Bartusch, M., Möhring, R. H., and Radermacher, F. J. (1988). Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16(1):199–240.

Baumann, P., Fündeling, C.-U., and Trautmann, N. (2015). The resource-constrained project scheduling problem with work-content constraints. In *Handbook on Project Management and Scheduling Vol. 1*, pages 533–544. Springer.

Ben-Tal, A., El Ghaoui, L., and Nemirovski, A. (2009). *Robust Optimization*, volume 28. Princeton University Press.

Ben-Tal, A., Goryashko, A., Guslitzer, E., and Nemirovski, A. (2004). Adjustable robust solutions of uncertain linear programs. *Mathematical Programming*, 99(2):351–376.

Ben-Tal, A. and Nemirovski, A. (1998). Robust convex optimization. *Mathematics of Operations Research*, 23(4):769–805.

Ben-Tal, A. and Nemirovski, A. (1999). Robust solutions of uncertain linear programs. *Operations Research Letters*, 25(1):1–13.

Ben-Tal, A. and Nemirovski, A. (2000). Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, 88(3):411–424.

Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252.

Bendotti, P., Chrétienne, P., Fouilhoux, P., and Pass-Lanneau, A. (2019). The anchor-robust project scheduling problem. Preprint hal-02144834.

Bertsimas, D., Brown, D. B., and Caramanis, C. (2011). Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501.

Bertsimas, D. and Caramanis, C. (2010). Finite adaptability in multistage linear optimization. *IEEE Transactions on Automatic Control*, 55(12):2751–2766.

Bertsimas, D. and Sim, M. (2003). Robust discrete optimization and network flows. *Mathematical programming*, 98(1):49–71.

Bertsimas, D. and Sim, M. (2004). The price of robustness. *Operations research*, 52(1):35–53.

Bianco, L. and Caramia, M. (2011). A new lower bound for the resource-constrained project scheduling problem with generalized precedence relations. *Computers & Operations Research*, 38(1):14–20.

Bianco, L. and Caramia, M. (2012). An exact algorithm to minimize the makespan in project scheduling with scarce resources and generalized precedence relations. *European Journal of Operational Research*, 219(1):73–85.

Bianco, L. and Caramia, M. (2013). A new formulation for the project scheduling problem under limited resources. *Flexible Services and Manufacturing Journal*, 25(1-2):6–24.

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

Bold, M. and Goerigk, M. (2021). A compact reformulation of the two-stage robust resource-constrained project scheduling problem. *Computers & Operations Research*, 130:105232.

Bold, M. and Goerigk, M. (2022a). Investigating the recoverable robust single machine scheduling problem under interval uncertainty. *Discrete Applied Mathematics*, 313:99–114.

Bold, M. and Goerigk, M. (2022b). Recoverable robust single machine scheduling with polyhedral uncertainty. *arXiv preprint arXiv:2011.06284v2*.

Bougeret, M., Pessoa, A. A., and Poss, M. (2019). Robust scheduling with budgeted uncertainty. *Discrete Applied Mathematics*, 261:93–107.

Brucker, P., Drexl, A., Möhring, R., Neumann, K., and Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European journal of operational research*, 112(1):3–41.

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

Bruni, M. E., Pugliese, L. D. P., Beraldi, P., and Guerriero, F. (2017). An adjustable robust optimization model for the resource-constrained project scheduling problem with uncertain activity durations. *Omega*, 71:66–84.

Bruni, M. E., Pugliese, L. D. P., Beraldi, P., and Guerriero, F. (2018). A computational study of exact approaches for the adjustable robust resource-constrained project scheduling problem. *Computers & Operations Research*, 99:178–190.

Buchheim, C. and Kurtz, J. (2016). Min-max-min robust combinatorial optimization subject to discrete uncertainty. *Optimization online.*

Buchheim, C. and Kurtz, J. (2017). Min-max-min robust combinatorial optimization. *Mathematical Programming*, 163(1-2):1–23.

Buchheim, C. and Kurtz, J. (2018). Robust combinatorial optimization under convex and discrete cost uncertainty. *EURO Journal on Computational Optimization*, 6(3):211–238.

Büsing, C. (2012). Recoverable robust shortest path problems. *Networks*, 59(1):181–189.

Büsing, C., Koster, A. M., and Kutschka, M. (2011). Recoverable robust knapsacks: the discrete scenario case. *Optimization Letters*, 5(3):379–392.

Cesta, A., Oddi, A., and Smith, S. F. (2002). A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8(1):109–136.

Chang, Z., Song, S., Zhang, Y., Ding, J.-Y., Zhang, R., and Chiong, R. (2017). Distributionally robust single machine scheduling with risk aversion. *European Journal of Operational Research*, 256(1):261–274.

Chassein, A. and Goerigk, M. (2016). On the recoverable robust traveling salesman problem. *Optimization Letters*, 10(7):1479–1492.

Chassein, A. and Goerigk, M. (2017). Minmax regret combinatorial optimization problems with ellipsoidal uncertainty sets. *European Journal of Operational Research*, 258(1):58–69.

Chassein, A. and Goerigk, M. (2021). On the complexity of min–max–min robustness with two alternatives and budgeted uncertainty. *Discrete Applied Mathematics*, 296:141–163.

Chassein, A., Goerigk, M., Kasperski, A., and Zieliński, P. (2018). On recoverable and two-stage robust selection problems with budgeted uncertainty. *European Journal of Operational Research*, 265(2):423–436.

Chassein, A., Goerigk, M., Kurtz, J., and Poss, M. (2019). Faster algorithms for min-max-min robustness for combinatorial problems with budgeted uncertainty. *European Journal of Operational Research*, 279(2):308–319.

Christofides, N., Alvarez-Valdés, R., and Tamarit, J. M. (1987). Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29(3):262–273.

Daniels, R. L. and Kouvelis, P. (1995). Robust scheduling to hedge against processing time uncertainty in single-stage production. *Management Science*, 41(2):363–376.

de Azevedo, G. H. I., Pessoa, A. A., and Subramanian, A. (2021). A satisfiability and workload-based exact method for the resource constrained project scheduling problem with generalized precedence constraints. *European Journal of Operational Research*, 289(3):809–824.

De Reyck, B. and Herroelen, W. (1998). A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research*, 111(1):152–174.

Debels, D. and Vanhoucke, M. (2007). A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Operations Research*, 55(3):457–469.

Deĭneko, V. G., Woeginger, G. J., et al. (2006). On the robust assignment problem under a fixed number of cost scenarios. *Operations Research Letters*, 34(2):175–179.

Delage, E. and Iancu, D. A. (2015). Robust multistage decision making. In *The operations research revolution*, pages 20–46. INFORMS.

Demeulemeester, E. L. and Herroelen, W. S. (1997). A branch-and-bound procedure for the generalized resource-constrained project scheduling problem. *Operations Research*, 45(2):201–212.

Demeulemeester, E. L. and Herroelen, W. S. (2006). *Project scheduling: a research handbook*, volume 49. Springer Science & Business Media.

Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213.

Dorndorf, U., Pesch, E., and Phan-Huy, T. (2000). A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. *Management Science*, 46(10):1365–1384.

Drezet, L.-E. and Billaut, J.-C. (2008). Employee scheduling in an IT company. In Artigues, C., Demassey, S., and Neron, E., editors, *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*, pages 243–255. ISTE/Wiley.

Edmonds, J. (1965). Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards*, 69B:125–130.

El Ghaoui, L. and Lebret, H. (1997). Robust solutions to least-squares problems with uncertain data. *SIAM Journal on matrix analysis and applications*, 18(4):1035–1064.

El Ghaoui, L., Oustry, F., and Lebret, H. (1998). Robust solutions to uncertain semidefinite programs. *SIAM Journal on Optimization*, 9(1):33–52.

Elsayed, S., Sarker, R., Ray, T., and Coello, C. C. (2017). Consolidated optimization algorithm for resource-constrained project scheduling problems. *Information Sciences*, 418:346–362.

Eufinger, L., Kurtz, J., Buchheim, C., and Clausen, U. (2020). A robust approach to the capacitated vehicle routing problem with uncertain costs. *INFORMS Journal on Optimization*, 2(2):79–95.

Fischer, D., Hartmann, T. A., Lendl, S., and Woeginger, G. J. (2020). An investigation of the recoverable robust assignment problem. *arXiv preprint arXiv:2010.11456*.

Franck, B., Neumann, K., and Schwindt, C. (2001). Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. *OR-Spektrum*, 23(3):297–324.

Fridman, I., Pesch, E., and Shafransky, Y. (2020). Minimizing maximum cost for a single machine under uncertainty of processing times. *European Journal of Operational Research*, 286(2):444–457.

Fündeling, C.-U. and Trautmann, N. (2010). A priority-rule method for project scheduling with work-content constraints. *European Journal of Operational Research*, 203(3):568–574.

Gabrel, V., Murat, C., and Thiele, A. (2014). Recent advances in robust optimization: An overview. *European journal of operational research*, 235(3):471–483.

Ghouila-Houri, A. (1962). Caractérisation des matrices totalement unimodulaires. *Comptes Redus Hebdomadaires des Séances de l'Académie des Sciences (Paris)*, 254:1192–1194.

Goerigk, M., Kasperski, A., and Zieliński, P. (2021a). Combinatorial two-stage minmax regret problems under interval uncertainty. *Annals of Operations Research*, 300(1):23–50.

Goerigk, M., Kasperski, A., and Zieliński, P. (2022a). Robust two-stage combinatorial optimization problems under convex second-stage cost uncertainty. *Journal of Combinatorial Optimization*, 43:497–527.

Goerigk, M., Kurtz, J., and Poss, M. (2020). Min–max–min robustness for combinatorial problems with discrete budgeted uncertainty. *Discrete Applied Mathematics*, 285:707–725.

Goerigk, M., Lendl, S., and Wulf, L. (2021b). On the recoverable traveling salesman problem. *arXiv preprint arXiv:2111.09691*.

Goerigk, M., Lendl, S., and Wulf, L. (2022b). Recoverable robust representatives selection problems with discrete budgeted uncertainty. *European Journal of Operational Research*.

Goerigk, M. and Schöbel, A. (2016). Algorithm engineering in robust optimization. In *Algorithm Engineering*, pages 245–279. Springer.

Gonçalves, J. F., Resende, M. G., and Mendes, J. J. (2011). A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. *Journal of Heuristics*, 17(5):467–486.

Gorissen, B. L., Yanıkoğlu, İ., and den Hertog, D. (2015). A practical guide to robust optimization. *Omega*, 53:124–137.

Gourgand, M., Grangeon, N., and Norre, S. (2008). Assembly shop scheduling. In Artigues, C., Demassey, S., and Neron, E., editors, *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*, pages 229–242. ISTE/Wiley.

Graham, R., Lawler, E., Lenstra, J., and Rinnooy Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. In *Annals of Discrete Mathematics*, volume 5, pages 287–326. Elsevier.

Hanasusanto, G. A., Kuhn, D., and Wiesemann, W. (2015). K-adaptability in two-stage robust binary programming. *Operations Research*, 63(4):877–891.

Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics (NRL)*, 45(7):733–750.

Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics (NRL)*, 49(5):433–448.

Hartmann, S. and Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of operational research*, 207(1):1–14.

Hartmann, S. and Briskorn, D. (2022). An updated survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 297(1):1–14.

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

Herroelen, W., De Reyck, B., and Demeulemeester, E. (1998). Resource-constrained project scheduling: A survey of recent developments. *Computers & Operations Research*, 25(4):279–302.

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

Hindi, K. S., Yang, H., and Fleszar, K. (2002). An evolutionary algorithm for resource-constrained project scheduling. *IEEE Transactions on evolutionary computation*, 6(5):512–518.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI.

Hradovich, M., Kasperski, A., and Zieliński, P. (2017a). Recoverable robust spanning tree problem under interval uncertainty representations. *Journal of Combinatorial Optimization*, 34(2):554–573.

Hradovich, M., Kasperski, A., and Zieliński, P. (2017b). The recoverable robust spanning tree problem with interval costs is polynomially solvable. *Optimization Letters*, 11(1):17–30.

Igelmund, G. and Radermacher, F. J. (1983a). Algorithmic approaches to preselective strategies for stochastic scheduling problems. *Networks*, 13(1):29–48.

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

Kasperski, A. (2005). Minimizing maximal regret in the single machine sequencing problem with maximum lateness criterion. *Operations Research Letters*, 33(4):431–436.

Kasperski, A. and Zieliński, P. (2008). A 2-approximation algorithm for interval data minmax regret sequencing problems with the total flow time criterion. *Operations Research Letters*, 36(3):343–344.

Kasperski, A. and Zieliński, P. (2011). On the approximability of robust spanning tree problems. *Theoretical Computer Science*, 412(4-5):365–374.

Kasperski, A. and Zielinski, P. (2014). Minmax (regret) scheduling problems. *Sequencing and scheduling with inaccurate data*, pages 159–210.

Kasperski, A. and Zieliński, P. (2016a). Robust discrete optimization under discrete and interval uncertainty: A survey. In *Robustness analysis in decision aiding, optimization, and analytics*, pages 113–143. Springer.

Kasperski, A. and Zieliński, P. (2016b). Single machine scheduling problems with uncertain parameters and the OWA criterion. *Journal of Scheduling*, 19(2):177–190.

Kasperski, A. and Zieliński, P. (2017). Robust recoverable and two-stage selection problems. *Discrete Applied Mathematics*, 233:52–64.

Kasperski, A. and Zieliński, P. (2019). Risk-averse single machine scheduling: complexity and approximation. *Journal of Scheduling*, 22(5):567–580.

Kim, J.-L. (2013). Genetic algorithm stopping criteria for optimization of construction resource scheduling problems. *Construction Management and Economics*, 31(1):3–19.

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

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

Kolisch, R. and Hartmann, S. (1999). Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In *Project scheduling*, pages 147–178. Springer.

Kolisch, R. and Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European journal of operational research*, 174(1):23–37.

Kolisch, R., Meyer, K., Mohr, R., Schwindt, C., and Urmann, M. (2003). Ablaufplanung fur die Leitstrukturoptimierung in der Pharmaforschung. *Zeitschrift für Betriebswirtschaft*, 73(8):825–848.

Kolisch, R. and Padman, R. (2001). An integrated survey of deterministic project scheduling. *Omega*, 29(3):249–272.

Kolisch, R. and Sprecher, A. (1997). PSPLIB - A project scheduling problem library: OR software - ORSEP operations research software exchange program. *European Journal of Operational Research*, 96(1):205–216.

Koné, O., Artigues, C., Lopez, P., and Mongeau, M. (2011). Event-based milp models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3–13.

Koné, O., Artigues, C., Lopez, P., and Mongeau, M. (2013). Comparison of mixed integer linear programming models for the resource-constrained project scheduling problem with consumption and production of resources. *Flexible Services and Manufacturing Journal*, 25(1):25–47.

Kouvelis, P. and Yu, G. (1997). *Robust discrete optimization and its applications.* Kluwer Academic Publishers Dordrecht, Netherlands.

Lamas, P. and Demeulemeester, E. (2016). A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations. *Journal of Scheduling*, 19(4):409–428.

Lebedev, V. and Averbakh, I. (2006). Complexity of minimizing the total flow time with interval data and minmax regret criterion. *Discrete Applied Mathematics*, 154(15):2167–2177.

Li, H. and Womer, N. K. (2015). Solving stochastic resource-constrained project scheduling problems by closed-loop approximate dynamic programming. *European Journal of Operational Research*, 246(1):20–33.

Liebchen, C., Lübbecke, M., Möhring, R., and Stiller, S. (2009). The concept of recoverable robustness, linear programming recovery, and railway applications. In *Robust and online large-scale optimization*, pages 1–27. Springer.

Lim, A., Ma, H., Rodrigues, B., Tan, S. T., and Xiao, F. (2013). New metaheuristics for the resource-constrained project scheduling problem. *Flexible Services and Manufacturing Journal*, 25(1):48–73.

Lu, C.-C., Lin, S.-W., and Ying, K.-C. (2012). Robust scheduling on a single machine to minimize total flow time. *Computers & Operations Research*, 39(7):1682–1691.

Lu, C.-C., Ying, K.-C., and Lin, S.-W. (2014). Robust single machine scheduling for minimizing total flow time in the presence of uncertain processing times. *Computers & Industrial Engineering*, 74:102–110.

Mastrolilli, M., Mutsanas, N., and Svensson, O. (2013). Single machine scheduling with scenarios. *Theoretical Computer Science*, 477:57–66.

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

Montemanni, R. (2007). A mixed integer programming formulation for the total flow time single machine robust scheduling problem with interval data. *Journal of Mathematical Modelling and Algorithms*, 6(2):287–296.

Naber, A. (2017). Resource-constrained project scheduling with flexible resource profiles in continuous time. *Computers & Operations Research*, 84:33–45.

Naber, A. and Kolisch, R. (2014). MIP models for resource-constrained project scheduling with flexible resource profiles. *European Journal of Operational Research*, 239(2):335–348.

NDA (2019). Nuclear Provision: the cost of cleaning up Britain's historic nuclear sites. https://www.gov.uk/government/publications/nuclear-provision-explaining-the-cost-of-cleaning-up-britains-nuclear-legacy/nuclear-provision-explaining-the-cost-of-cleaning-up-britains-nuclear-legacy. Accessed: 12/06/2020.

Neumann, K. and Zhan, J. (1995). Heuristics for the minimum project-duration problem with minimal and maximal time lags under fixed resource constraints. *Journal of Intelligent Manufacturing*, 6(2):145–154.

Pass-Lanneau, A., Bendotti, P., and Brunod-Indrigo, L. (2020). Exact and heuristic methods for Anchor-Robust and Adjustable-Robust RCPSP. *arXiv preprint arXiv:2011.02020*.

Pellerin, R., Perrier, N., and Berthaut, F. (2020). A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 280(2):395–416.

Pereira, J. and Averbakh, I. (2011). Exact and heuristic algorithms for the interval data robust assignment problem. *Computers & Operations Research*, 38(8):1153–1163.

Pritsker, A. A. B., Waiters, L. J., and Wolfe, P. M. (1969). Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16(1):93–108.

Proon, S. and Jin, M. (2011). A genetic algorithm with neighborhood search for the resource-constrained project scheduling problem. *Naval Research Logistics (NRL)*, 58(2):73–82.

Ranjbar, M. and Kianfar, F. (2010). Resource-constrained project scheduling problem with flexible work profiles: a genetic algorithm approach. *Transaction E: Industrial Engineering*, 17:25–35.

Schramme, T. (2014). *Modelle und Methoden zur Lösung des ressourcenbeschränkten Projektablaufplanungsproblems unter Berücksichtigung praxisrelevanter Aspekte*. PhD thesis, Universität Paderborn.

Schrijver, A. (2003). *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media.

Schutt, A., Feydy, T., Stuckey, P. J., and Wallace, M. G. (2013). Solving RCPSP/max by lazy clause generation. *Journal of Scheduling*, 16(3):273–289.

Schwindt, C. (1996). Generation of resource-constrained project scheduling problems with minimal and maximal time lags. Technical report, Institut für Wirtschaftstheorie und Operations Research, Univerität Karlsruhe.

Schwindt, C., Zimmermann, J., et al. (2015). *Handbook on project management and scheduling*. Springer.

Soyster, A. L. (1973). Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations research*, 21(5):1154–1157.

Subramanyam, A., Gounaris, C. E., and Wiesemann, W. (2020). K-adaptability in two-stage mixed-integer robust optimization. *Mathematical Programming Computation*, 12(2):193–224.

Tadayon, B. and Smith, J. C. (2015). Algorithms and complexity analysis for robust single-machine scheduling problems. *Journal of Scheduling*, 18(6):575–592.

Talbot, F. B. (1982). Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management science*, 28(10):1197–1210.

Thomas, D. J. (2015). *Matching Problems with Additional Resource Constraints*. PhD thesis, Universität Trier.

Tritschler, M., Naber, A., and Kolisch, R. (2017). A hybrid metaheuristic for resource-constrained project scheduling with flexible resource profiles. *European Journal of Operational Research*, 262(1):262–273.

Valls, V., Ballestin, F., and Quintanilla, S. (2008). A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European journal of operational research*, 185(2):495–508.

Valls, V., Ballestın, F., and Quintanilla, S. (2005). Justification and rcpsp: A technique that pays. *European Journal of Operational Research*, 165(2):375–386.

Vanhoucke, M. (2006). Scheduling an R&D project with quality-dependent time slots. In Gavrilova, M., Gervasi, O., Tan, C. J. K., Taniar, D., and Laganá, A., editors, *International Conference on Computational Science and Its Applications - ICCSA 2006*, volume 3982 of *Lecture Notes in Computer Science*, pages 621–630. Springer.

Wang, H., Li, T., and Lin, D. (2010). Efficient genetic algorithm for resource-constrained project scheduling problem. *Transactions of Tianjin University*, 16(5):376–382.

Weglarz, J. (2012). *Project scheduling: recent models, algorithms and applications*, volume 14. Springer Science & Business Media.

Węglarz, J., Józefowska, J., Mika, M., and Waligóra, G. (2011). Project scheduling with finite or infinite number of activity processing modes–a survey. *European Journal of operational research*, 208(3):177–205.

Yang, J. and Yu, G. (2002). On the robust single machine scheduling problem. *Journal of Combinatorial Optimization*, 6(1):17–33.

Yanıkoğlu, İ., Gorissen, B. L., and den Hertog, D. (2019). A survey of adjustable robust optimization. *European Journal of Operational Research*, 277(3):799–813.

Zamani, R. (2013). A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem. *European Journal of Operational Research*, 229(2):552–559.

Zeng, B. and Zhao, L. (2013). Solving two-stage robust optimization problems using a column-and-constraint generation method. *Operations Research Letters*, 41(5):457–461.

Zhao, H., Zhao, M., et al. (2010). A family of inequalities valid for the robust single machine scheduling polyhedron. *Computers & Operations Research*, 37(9):1610–1614.

Zhu, G., Bard, J. F., and Yu, G. (2007). A two-stage stochastic programming approach for project planning with uncertain activity durations. *Journal of Scheduling*, 10(3):167–180.

Zimmermann, A. (2016). An MIP-based heuristic for scheduling projects with work-content constraints. In *2016 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 1195–1199. IEEE.