# Bi-Objective Strategic and Operational Decision-Making in Redundancy Allocation Problems with Dynamic Maintenance

Luke Fairley, B.Sc.(Hons), M.Res.


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

November 2025

STOR-i
excellence with impact

# Abstract

Deterministic optimisation considers problems in which a decision-maker makes a single strategic decision, which are commonly formulated as mixed-integer linear programmes (MILPs). In contrast, stochastic dynamic optimisation considers problems involving sequential operational decisions in a random environment, typically modelled as Markov decision processes (MDPs). Separate still is bi-objective optimisation, in which trade-offs between competing objectives are explored. Frameworks that integrate strategic and operational decisions through MILPs and MDPs can be known as MDP Design frameworks, and the existing literature on this is limited to two prior studies each with their own approach, highlighting the complexity of merging these two disparate types of decision making. Furthermore, no previous attempts have been made to introduce multiple objectives into MDP Design. This thesis lies at the intersection of deterministic, stochastic dynamic, and bi-objective optimisation, seeking to answer questions as to how to formalise such problems, how to optimise such problems either exactly or approximately, and how the resulting modelling framework and solution methodologies can be applied in the context of redundancy allocation, i.e. the allocation of copies or backups of components in some system to improve its reliability, and the operational decisions as to how to maintain such systems to balance between reliability and costs.

This thesis provides a novel MDP Design framework that is well-suited to modelling more complex structural relationships between design decisions and the resulting state and action spaces of the MDP. Any problem within this framework is a MILP, and

can therefore be solved exactly by a MILP solver. However, this is known to be computationally expensive, so approximate solution methodologies are required to solve larger problems. A common theme throughout this thesis is the use of decomposition to break problems up into smaller subproblems, either breaking the strategic and operational stages of the MDP Design problem back into two distinct phases, or breaking a large and specially structured MDP down into smaller MDPs. The computational results show that these methods are effective, with a massive improvement in terms of scalability to larger problem instances, and solution quality comparable to that of the exact solver on smaller instances.

The results of these experiments demonstrate that solving problems with the proposed MDP Design framework is not beyond the current capabilities of Operational Research. As such, especially when paired with the structural flexibility of the framework, it is believed that these modelling and solution techniques could be suited to the integration of problems outside of system design and maintenance.

# Acknowledgements

First, I would like to thank my supervisors Rob Shone and Peter Jacko from Lancaster University, and Jefferson Huang from Naval Postgraduate School. Your consistent support during the course of my studies has made what could have been a very bumpy journey all the more smooth. I would especially like to thank the three of you for the academic freedom you have offered in not only allowing me, but encouraging me, to pursue the directions that I found interesting as the research took on a life of its own (and deviated heavily from the original research proposal). Thank you also for sharing in the frustration of, and supporting me through, some of the submission processes as we pursued publications; I never once felt that I was alone in believing that particular situations may have been taking the mickey.

I cannot go any further without thanking everyone at the STOR-i Centre for Doctoral Training. First and foremost, I would like to thank the administrative team, Nicky, Wendy, Kim, and Keilah, without whom STOR-i couldn't possibly run so smoothly. I can't possibly list everyone here, but I would like to thank: Stan Tendijck for his involved and encouraging supervision during my time as a STOR-i intern during COVID; the leadership team for keeping STOR-i alive and constantly evolving and improving; the "older" cohorts for making STOR-i a welcoming place to join and become a part of; my own cohort for the sense of solidarity throughout a busy Masters programme; and the "younger" cohorts for continuing to foster a positive sense of community. I'd especially like to thank my fellow MDP researcher Ben Lowery, who I could always rely

on for a good chat about anything.

Outside of STOR-i but still based in Lancaster, I'd like to thank my friends (in first name alphabetical order, no fighting!) Charlie Wells, Harry Stephenson, Jakub Waniek (who has self-deprecatingly insisted on being referred to as "leading by counterexample" on how to do a PhD), and Louise Frankland for the roughly-weekly board game nights throughout the PhD years. Those evenings were always (and continue to be) something to look forward to.

Turning to family, I can't possibly thank my Mum enough for everything she's done over the years, but to give a limited list I thank her for always being someone to turn to and an ear to bend over the trials and tribulations of the peer review process, for being a good daily rival in NYT Games, and for being an excellent free taxi. I'd also like to thank my Dad for our catch-up afternoons and evenings out in town and the ensuing free (for me) beer and food.

Finally, I'd like to thank my partner Carli who, as well as being another ear to bend PhD-wise, has supported me through my personal life. Notably, she set a far better example for keeping a healthy daily routine, compared to my previous late sleeps and long lie-ins.

# Declaration

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

Chapter 4 has been accepted for publication as Luke Fairley, Rob Shone, Jefferson Huang, Peter Jacko "A Bi-Objective Markov Decision Process Design Approach to Redundancy Allocation with Dynamic Maintenance for a Parallel System" in *European Journal of Operational Research*.

Chapter 5 is under submission as Luke Fairley, Rob Shone, Jefferson Huang, Peter Jacko "Decomposable Impulsive Markov Decision Processes with Multiple Objectives: Theory and Application".

This thesis is approximately 65449 words, including everything from the title through to appendices.

Luke Fairley

# Contents

# List of Figures

# List of Tables

# List of Symbols

| | |
|---|---|
| $\mathbb{E}\cdot$ | Expectation |
| $\mathbb{P}\cdot$ | Probability |
| $\mathbb{I}\cdot$ | Indicator function |
| $\mathbb{N}$ | Natural numbers starting from 1 |
| $\mathbb{Z}_{\geq 0}$ | Integers greater than or equal to zero (same as above) |
| $\mathbb{R}$ | Real numbers |
| $\{\cdot\}_{t\in\mathcal{T}}$ | $\mathcal{T}$-indexed sequence |
| $(\cdot)_{t\in\mathcal{T}}$ | $\mathcal{T}$-indexed vector |
| $\lvert\cdot\rvert$ | Cardinality |
| $[\cdot]$ | Integers up to, starting from 1 |
| $[\cdot,\cdot]$ | Closed Interval |
| $\bigtimes$ | Cartesian product |
| $.^T$ | Transpose |
| Bold, i.e. $\boldsymbol{a}, \boldsymbol{b}, ...$ | Vectors and tuples |
| Calligraphic, i.e. $\mathcal{A}, \mathcal{B}...$ | Sets |
| Fraktur, i.e. $\mathfrak{C}, \mathfrak{D}...$ | (Continuous or Discrete time) Markov decision processes |

# Chapter 1

# Introduction

Real-world systems, such as those found in manufacturing, power generation, electronics and so on, are usually made up of a number of interlinked components operating together. Of these components, some can be identified as *critical*, meaning if that component fails, the whole system ceases to function. One method to improve the overall reliability of such a system is to install some number of backup copies of these critical components, otherwise known as *redundancy*. This notion leads to a number of questions, the first of which is how we design such a system.

For each function to be served within the system, there may be a range of different components which serve that function, but which differ in other aspects such as costs or reliability. With redundancy in mind, we would want to determine which type (or types, if they can be used together) of component should be used for each function, and how many redundant backups of each component should be installed. Due to the large number of combinations of components that can be used together to specify an overall system design, this falls into the area of *combinatorial optimisation*.

Another question is that of maintenance: if we have lots of backups, do we necessarily need to perform maintenance on failed components straight away? If not, we can potentially save maintenance costs over the lifetime of the system, without necessarily

compromising too much on the system's overall reliability. As component failures are best modelled as happening at random, and these maintenance decisions will be made repeatedly and sequentially over the lifetime of the system, this falls into the camp of *stochastic dynamic optimisation*, an area which has quite different underlying mathematics and solution techniques than combinatorial optimisation. Finally, there is the question of how we balance between different priorities. Of course, we wish for our system to be very reliable, but small sacrifices in reliability due to design or maintenance decisions could lead to large long-run cost savings, and there is a balance to be struck between these two objectives. This concept is known as *bi-objective optimisation*, and this notion impacts on the techniques applied for both combinatorial and stochastic dynamic optimisation.

The goal of this thesis is to explore problems, models, solution methods, and algorithms at the intersection of combinatorial, dynamic, and bi-objective optimisation, and to do so in the context of systems with redundancy. Section 1.1 provides further exploration of some of the key themes found throughout this thesis, and Section 1.2 provides an outline of the content of each chapter.

## 1.1 Themes

### 1.1.1 Markov Decision Process Design

Decision-makers across many industries face problems which can broadly be characterised as strategic and operational, with tactical sitting in between. Strategic decisions are large one-off decisions whose impact will be felt over many years or decades. As such, they broadly fit into deterministic decision-making frameworks, such as *mixed integer linear programming* (MILP). On the other hand, operational decisions are small yet impactful decisions that are taken on a more frequent basis, fitting into the setting of sequential decision-making under uncertainty, often modelled as *Markov decision*

*processes* (MDPs). Seldom seen in the literature are attempts to combine these two levels of decision-making into integrated models that account for the interaction between those levels; in other words, static strategic decisions should account for the fact that the dynamic operational level will also be optimised.

A new and emerging framework that integrates two levels of decision-making is the MDP *Design* problem, as first coined by Brown et al. (2024), where first-stage strategic decisions are made to design some system or facility, and second-stage operational decisions are taken to manage the day-to-day running of the system or facility. To the best of our knowledge, only three prior works have offered models for the MDP Design problem, both of which have benefits and drawbacks. We offer a full exploration of these models in Chapter 3. The need to make design decisions that take into account dynamic system control, as well as the relative inflexibility of pre-existing works on MDP Design, provides the motivation to define a more flexible framework. Additionally, the unique difficulty and novelty of such a problem motivates the construction of novel solution methodologies.

### 1.1.2 Decomposition Methods

Decomposition-based algorithms are exact or approximate solution methods that break larger problems with some special structure down into multiple smaller sub-problems. They are particularly well-used in the context of MILPs, which are known to be an NP-Hard class of problems. As such, decomposition into smaller problems can greatly help in obtaining solutions in a reasonable time. Two common cases where decomposition can be employed are: where a problem is comprised of multiple stages (i.e., one can easily imagine that some decisions are taken first, then others strictly afterwards in response to those first decisions); and where a problem is comprised of multiple problems that to a great extent can be considered separate, but which must jointly satisfy some number of constraints. The former case is often tackled using a method called Benders

decomposition (Benders, 1962), which in turn makes use of *row generation* (i.e., starting a problem with a limited subset of the original constraints, and adding them only when they are needed); and the latter case is often tackled using Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960), which in turn makes use of *column generation* (i.e., starting with a limited subset of the original variables, and adding them if they improve the objective value).

While decomposition-based approached are more commonly associated with MILPs, they can also be applied to high-dimensional MDPs with a suitable structure. Two examples of such MDPs seen throughout the literature are the restless bandits of Whittle (1988) and the weakly-coupled MDPs of Meuleau et al. (1998). We continue the exploration of decomposition-based methods for MDPs and MDP Design problems throughout this thesis, in order to tackle the otherwise intractable nature of the problems we encounter.

### 1.1.3 Redundancy Allocation Problem

The Redundancy Allocation Problem (RAP) is a well-studied optimisation problem in the fields of reliability engineering and operational research. It is based on the idea that the reliability of a system made up of a number of interconnected components can be improved by the installation of redundant copies of some of those components. As such, when one copy fails, the system can simply switch to using another copy, allowing for the system to continue functioning uninterrupted. Only if all copies of a component are non-functional, or if the switching mechanism itself fails, would we see a disruption in the functionality of the overall system. These copies may serve the same purpose but have different attributes such as cost, weight, efficiency, and so on. The RAP is then the problem of determining how many copies of each type of component to be installed with respect to certain constraints and objectives.

Of interest to us are versions of the RAP where the components are repairable,

and operational costs are of concern. If operational costs are not of any concern and we need only optimise system reliability, then it is clear that any component should be repaired as soon as it fails, no matter the repair cost or the remaining number of components. However, if operational costs are of concern, then this raises the question as to when repairs should be performed on the system. This reflects the real-world cost-benefit trade-off between the cost of maintenance and the impact on system reliability. In this thesis, we limit our investigation to cases where the system design is simple, i.e., either one component or components connected in series, as opposed to complicated networks. This respectively leads to designs which are *parallel* systems, i.e., systems where all components serve the same purpose, and the system only fails if all components have failed simultaneously; and series-parallel systems, which are systems comprised of multiple parallel *subsystems* in series, meaning the overall system fails if only one of these subsystems has failed. This offers an important first step towards such design-maintenance optimisation for those more complicated systems.

The joint consideration of operational costs and system reliability throughout this manuscript motivates the use of MDPs to model the sequential maintenance problem for redundant systems, and the design of such systems with the knowledge that they will be dynamically maintained further motivates the development of both a framework for the MDP Design problem, and a solution methodology for the resulting framework.

## 1.2 Thesis Outline

The remainder of the thesis is organised as follows:

- Chapter 2 covers the basics of linear programming, mixed-integer programming, MDPs, and bi-objective optimisation.
- Chapter 3 investigates the two previous works on the MDP Design problem, analysing their benefits and limitations, and presents our own framework for this

problem. We additionally show that our and previous frameworks all have standard MDP counterparts.

- Chapter 4 investigates the integration of a single-subsystem (i.e., parallel) RAP with dynamic maintenance considering two objectives: operational costs and reliability. We formulate this problem with our MDP Design framework, provide a general heuristic solution methodology for such problems, and investigate the performance of a heuristic derived from this methodology on our problem. We also explore how varying attributes of the model, such as component-wise usage and repair costs, affects the space of solutions.

- Chapter 5 investigates the notions of impulsivity (i.e., instantaneous transitions) and decomposability in continuous-time MDPs. This theoretical chapter develops a comparatively simple (albeit less feature-rich) framework for impulsivity compared to previous literature, extends it to a decomposable model, then provides a decomposition-based heuristic for solving such problems. We apply this modelling framework and solution methodology to series-parallel systems, which is a type of system that is common in the RAP literature.

- Chapter 6 provides work towards the fusion of Chapter 4 and Chapter 5, considering the joint design and maintenance optimisation of a series-parallel system. We provide a detailed outline of the optimisation model and a column-generation-based heuristic with the potential for many extensions. The computational study section provides some preliminary results which explore the extent to which this model can be solved exactly using a standard MILP solver, and explores partial implementation of our solution methodology.

- Chapter 7 summarises the main findings and limitations from the research presented in this thesis, and discusses some directions for further research.

## 1.3 Statement of Contributions

The vast majority of the contributions of this thesis were conceptualised, implemented, and written independently by myself, with guidance, written revisions, and encouragement from my supervisors - Rob, Peter, and Jefferson - along the way. The original project proposed by the supervisors was a single-objective MDP problem regarding the maintenance of a network controlled by a min-cost flow problem. We originally considered a simplified version of this problem where a network was comprised of one source node, one destination node, and any number of heterogeneous links between the two. There was a natural connection between this type of network and the parallel subsystems that arise from redundancy allocation problems. The decision to pivot to the redundancy allocation literature, the focus on parallel and series-parallel systems, the integration of design variables to obtain the MDP Design problem, and the conversion to a bi-objective problem were all my decision. The primary research contributions are found in Chapter 3, Chapter 4, Chapter 5, and Chapter 6. This introductory chapter, the following preliminaries chapter, and the concluding Chapter 7 are all written by myself, with revisions suggested by my supervisors.

In Chapter 3, it is worth noting that I was first alerted to the existence of Brown et al. (2024) by Jefferson soon after it was published, and I was first notified of the existence of the PhD thesis of Liu (2022) by Peter. Aside from this, the rest of this chapter, notably the new MDP Design framework and the equivalences to standard MDPs, are my own work.

In Chapter 4, as previously noted, the choice to pursue redundancy allocation, MDP Design, and a bi-objective formulation were all my own. With respect to methodology, the exact formulation and implementation of the MDP Design problem, and the APP methodology, are my own contributions. I was first alerted of the probability chains paper (O'Hanley et al., 2013) by Peter when I was trying to approximately linearise the design problem, but from there I recognised its applicability myself and pursued its

implementation. The use of impulsive control for modelling the repair actions was also suggested by Peter. The use of the long-run average cost criterion was first suggested by Rob in the very early days of the project; however, after swapping to a bi-objective formulation, I noted its usefulness in having a long-run average reliability objective, and in avoiding issues of choosing a discount factor for such a sensitive objective. The use of linear programming (LP) to solve MDPs was first suggested by Jefferson following some convergence issues with standard DP methods, and was later pursued when LP become necessary to solve the MDP Design model. All implementation required for the computational study was my own code, with some aspects of the experiments conducted resulting from conversations with supervisors.

In Chapter 5, the idea to use Dantzig-Wolfe decomposition on a multi-objective CTMDP with special structure was my own, inspired by the use of decomposition for series-parallel RAPs by Reihaneh et al. (2022). I also recognised the need for, and developed, the necessary theory for impulsive and decomposable-impulsive CTMDPs to allow me to use such a method. The implementation and experimental design, including the development of REPOG to measure Pareto-suboptimality, was also my own. Rob aided by reading through this work in detail, ensuring that the theoretical contributions were mathematically sound. Peter helped in restructuring this piece of work, notably suggesting the use of a structure in which technical details can be easily skipped over by a reader who is just interested in the modelling framework, as adopted in Section 5.2. Again, all three supervisors assisted by suggesting revisions to the original draft.

Chapter 6 continues to demonstrate independent contributions. The method used to efficiently construct the state-action space, the use of Dantzig-Wolfe decomposition, the formulation of pricing subproblems and their conversion into bi-objective problems, and all other aspects of the methodology are my own. All implementation and experiments are also my own work. All three supervisors again assisted in reviewing and revising this chapter.

# Chapter 2

# Preliminaries

In this chapter, we introduce the preliminary models and methods that will be used in subsequent chapters. In Section 2.1 we introduce deterministic optimisation, which is a framework used to model large one-off decisions, such as system design. In Section 2.2, we introduce the Markov decision process, which is a framework used for sequential decision making under uncertainty. Finally, in Section 2.3, we introduce bi-objective optimisation, which extends both optimisation models to allow for the efficient balancing of two objectives, as opposed to the optimisation of just one. This chapter serves as a review of relevant theory and methods in general; a review of the literature on the redundancy allocation problem specifically can be found in Section 4.1.1.

## 2.1 Deterministic Optimisation

In this section, we introduce the aspects of deterministic optimisation which are relevant to our work. In all generality, a deterministic optimisation problem can be written as:

$$\min_{x \in \mathcal{X}} f(x),$$

where $\mathcal{X}$ is some (closed) set (so "min" is well defined), $f : \mathcal{X} \to \mathbb{R}$ is a function from the set to the real numbers, and the objective is to find an element $x^* \in \mathcal{X}$ such that $f(x^*) \leq f(x)$ for all $x \in \mathcal{X}$. Throughout, we focus on continuous linear and mixed-integer linear programming problems. In Section 2.1.1 we introduce continuous linear programming (often shortened to just linear programming, or LP) where we take the decision space $\mathcal{X}$ to be a subset of $\mathbb{R}^n$ with specific properties, and in Section 2.1.2 we introduce mixed-integer linear programming, where we take $\mathcal{X}$ to be a subset of $\mathbb{Z}^m \times \mathbb{R}^n$, i.e., some of the decision variables are restricted to being pure integer values. In this context, we understand elements of $\mathcal{X}$ to be vectors, and therefore write them as $\boldsymbol{x}$. A thorough introduction to LP can be found in Karloff (2008), and an introduction to MILPs can be found in Wolsey (2020).

### 2.1.1 Continuous Linear Programming

A linear programming problem is a deterministic optimisation problem that can be written in the *standard form* (following the terminology of Karloff (2008), this language can differ between authors):

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \left\{ \boldsymbol{c}^T \boldsymbol{x} : A\boldsymbol{x} = \boldsymbol{b}, x_i \geq 0 \text{ for all } i \right\},$$

where $n$ is a number of decision variables, $\boldsymbol{c} \in \mathbb{R}^n$ is a cost vector, $A \in \mathbb{R}^{m \times n}$ is a matrix of constraint contributions with linearly independent columns (if this is not the case, such columns/variables can be eliminated) and independent rows (if this is not the case, such rows/constraints can be eliminated), and $\boldsymbol{b} \in \mathbb{R}^m$ is a vector of constraint values. While this may seem restrictive, equality constraints can always be obtained from inequality constraints by adding slack variables (i.e., $\boldsymbol{a}^T\boldsymbol{x} \leq \boldsymbol{b}$ becomes $\boldsymbol{a}^T\boldsymbol{x} + s = \boldsymbol{b}$ for some slack variable $s \geq 0$, and likewise $\boldsymbol{a}^T\boldsymbol{x} \geq \boldsymbol{b}$ becomes $\boldsymbol{a}^T\boldsymbol{x} - s = \boldsymbol{b}$), and a "free" variable $x \in \mathbb{R}$ can be split into its positive and negative components, i.e.,

$x = x^+ - x^-$ where $x^+, x^- \geq 0$.

The equation $A\boldsymbol{x} = \boldsymbol{b}$ with $\boldsymbol{x} \geq 0$ (with $\geq$ taken component-wise) represents a polytope embedded in a high-dimensional space which may in general be unbounded; however, we need only consider the bounded case for the purposes of this thesis (and for brevity). If this equation has no solutions (i.e., the polytope is empty), then we call the problem *infeasible*. Otherwise, the polytope has vertices that correspond to *basic feasible solutions*, which are solutions $\boldsymbol{x}_B = B^{-1}\boldsymbol{b}$ for invertible square submatrices $B$ of $A$ with all $m$ rows (all constraints must be satisfied) and $m$ columns (we need at most one variable per constraint to satisfy the constraint matrix). It is known that there always exists a basic feasible solution that is optimal. Retroactively, we may call this LP the *primal* LP (PLP), and define the *dual* LP (DLP) as follows:

$$\max_{\boldsymbol{y} \in \mathbb{R}^m} \left\{ \boldsymbol{b}^T \boldsymbol{y} : A^T \boldsymbol{y} \leq \boldsymbol{c}, y_i \in \mathbb{R} \right\},$$

where we interpret this problem to be the one where the roles of the variables and constraints of PLP have "swapped". By a property known as *strong duality*, optimal solutions $\boldsymbol{x}^*$ for the PLP and $\boldsymbol{y}^*$ for the DLP satisfy the equation:

$$\boldsymbol{c}^T \boldsymbol{x}^* = \boldsymbol{b}^T \boldsymbol{y}^*.$$

The optimal dual variables $\boldsymbol{y}^*$ can be roughly understood as the instantaneous rate of change of the optimal objective function with respect to the right hand side (RHS) values of the constraints.

LPs can be solved exactly by three main methods: primal simplex, dual simplex, and interior point (also known as the "barrier" method). All three methods are often implemented in off-the-shelf LP solvers (we use Gurobi (Gurobi Optimization, LLC, 2024) throughout this work). We need not fully explain each method here; however, some understanding of the primal simplex method is desirable in order to understand

the methods in Section 2.1.3 and Section 2.1.4. The primal simplex method works by iteratively updating a basic feasible solution to move towards optimality. At each iteration, given an invertible submatrix $B$ of $A$ that we call the *basis* which defines the incumbent solution $\boldsymbol{x}_B = B^{-1}\boldsymbol{b}$, the method identifies an *entering variable*, which is a "promising" column $s$ of $A$ not currently in the basis $B$; and a *leaving variable*, which is the first variable $r$ to reach 0 as the entering variable $x_s$ increases from 0. The basis $B$ is updated to include column $s$ and exclude column $r$ efficiently by using methods from linear algebra. Our interest with respect to Section 2.1.3 is the determination of the entering variable. Let $N$ be the submatrix of $A$ with all of the rows, and all of the columns not included in $B$, so that $N$ is the submatrix of $A$ corresponding to the non-basic variables. We can write the following:

$$\boldsymbol{c}^T\boldsymbol{x}_B = \boldsymbol{c}_B^T B^{-1}\boldsymbol{b} + (\boldsymbol{c}_N - \boldsymbol{c}_B^T B^{-1}N)\boldsymbol{x}_N,$$

where $(\boldsymbol{c}_N - \boldsymbol{c}_B^T B^{-1}N)$ is the *reduced cost vector* of the non-basic variables. The entering variable is that with the minimum negative reduced cost, i.e., the lowest entry in the reduced cost vector, or the non-basic variable with the greatest instantaneous rate of decrease in the objective function. If all entries are positive, then we terminate the process as optimality has been achieved. It is enlightening to take a 'dual' perspective to this process. Consider the PLP with matrix A restricted to $B$ (so it only has one solution, which is automatically the optimal solution to this problem): $\min_{\boldsymbol{x}_B}\left\{\boldsymbol{c}_B^T\boldsymbol{x}_B : B\boldsymbol{x}_B = b, x_{B_i} \geq 0\right\}$, and consider the dual of this problem $\max_{\boldsymbol{y}}\left\{\boldsymbol{b}^T\boldsymbol{y} : B^T\boldsymbol{y} \leq \boldsymbol{c}_B\right\}$, then by strong duality we have:

$$\begin{aligned}
&\boldsymbol{c}_B^T\boldsymbol{x}_B = \boldsymbol{b}^T\boldsymbol{y}^* \\
\implies &\boldsymbol{c}_B^T B^{-1}\boldsymbol{b} = \boldsymbol{y}^{*T}\boldsymbol{b} \text{ (expand out LHS and swap around RHS)} \\
\implies &\boldsymbol{c}_B^T B^{-1} = \boldsymbol{y}^{*T}.
\end{aligned}$$

As such, the reduced cost $rc$ of a non-basic variable $i$ can be expressed as $rc(i) = c_i - \boldsymbol{y}^{*T} A_i$, where $c_i$ is simply the $i^{\text{th}}$ entry of the cost vector and $A_i$ is the $i^{\text{th}}$ column of $A$ (i.e., the contribution of variable $i$ to each constraint). The problem of finding an entering variable itself can then be thought of as a small optimisation problem $\min_i\{c_i - \boldsymbol{y}^{*T} A_i\}$, where $i$ runs over the indices of the non-basic variables. Normally, this small problem is solved by simply running over all $i$ and finding the best value; however, in special cases where there are too many values of $i$ to look over explicitly but there is some additional structure to exploit, this problem becomes more interesting. In general, the method of starting with a small, feasible subset of the variables of an LP and then iteratively adding variables by solving the reduced-cost minimisation problem is called *column generation*, and this will be explored in Section 2.1.3. This concludes all necessary details for continuous LPs, so we now move on to mixed-integer linear programming.

### 2.1.2 Mixed Integer Linear Programming

A mixed-integer LP (MILP) is a deterministic optimisation problem similar to a standard LP, except with the additional constraint that a predetermined set of the decision variables can only take integer values. This can be formalised as:

$$\min_{\boldsymbol{x} \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2}} \left\{ \boldsymbol{c}^T \boldsymbol{x} : A\boldsymbol{x} = \boldsymbol{b} \right\}.$$

MILPs are a more flexible and far more difficult class of optimisation problems, famously being NP-hard in general. Solution methods often make use of the *linear relaxation* of the MILP, given as $\min_{\boldsymbol{x} \in \mathbb{R}^{n_1+n_2}} \left\{ \boldsymbol{c}^T \boldsymbol{x} : A\boldsymbol{x} = \boldsymbol{b} \right\}$. This is almost always an easier problem to solve, but in general returns infeasible solutions to the MILP, meaning some of the variables that should be integer are instead fractional. Exact methods for solving a MILP try to "tighten" the linear relaxation to remove fractional solutions. One such

method is to use *valid inequalities*, which are inequalities that can be added to the linear relaxation of a MILP which remove some fractional solutions, but do not remove any integer solutions. Such inequalities are also known as *cuts* or cutting planes. These cuts may be generated by analysing the problem structure and parameters beforehand and adding them in bulk, or by repeatedly re-optimising the linear relaxation and "cutting off" the solution found if it is integer-infeasible (where determining this cut is known as a *separation problem*). An algorithm that solves a MILP just using valid inequalities is called a cutting planes algorithm.

The other main method that makes use of linear relaxations to solve MILPs is *branch-and-bound*, or BB. A BB algorithm works by repeatedly solving linear relaxations, identifying a fractional variable that should be integral (say $x_i = \overline{x_i} \notin \mathbb{Z}$), and generating two new problems: one with a new constraint $x_i \leq \lfloor \overline{x_i} \rfloor$, and another with constraint $x_i \geq \lceil \overline{x_i} \rceil$; this process is known as *branching*. A linear relaxation of the MILP equipped with some set of branching inequalities of the form described is known as a *node*, and the BB method maintains a list of unsolved nodes to be checked. The BB method also maintains an upper bound solution, which is always the best integer feasible solution to the MILP found so far. This can be identified during the BB process by branching alone or by getting lucky with a linear relaxation, or by adding in some additional heuristics. This upper bound is important for ignoring nodes with no potential. For any node, we have a number of cases:

1. Infeasible - in this case the node is *fathomed* (i.e., deleted with no further branching);

2. Solved and integer - in this case the node provides a feasible solution to the MILP, and the objective value can be used to update the upper bound if it is an improvement, then the node is fathomed;

3. Solved and above upper bound - in this case the linear relaxation (regardless of

whether or not it is integer) is worse than the current upper bound, so no further branching is needed and the node can be fathomed;

4. Solved, fractional, and below upper bound - this is the case where we must branch on one of the fractional variables, producing two new nodes.

A BB method repeatedly selects nodes from the list as described and deals with them as above, and does so until the list is empty, at which point the optimal solution has been found (or the problem is found to be infeasible). This leaves two questions: how do we choose nodes from the list, and how do we choose which variable to branch on? Answers to these questions can very much be problem dependent, and as such are outside the scope of this work. This basic overview of BB is sufficient background for the discussion of a BB-based method mentioned in Section 2.1.4. A final note is that cutting planes and branch-and-bound can be combined into a method known as *branch-and-cut*, which adds cuts at each of the nodes to strengthen the relaxation, increasing the chance that the solution found will be integer. This concludes the necessary knowledge for general MILPs and for deterministic optimisation, so we may now discuss decomposition methods for a special class of LPs and MILPs.

### 2.1.3 Dantzig-Wolfe Decomposition for Continuous LPs

*Dantzig-Wolfe decomposition* is a method for solving LPs with a *block-diagonal structure*, first introduced by Dantzig and Wolfe (1960). We begin with the analysis of a polytope $A\boldsymbol{x} = \boldsymbol{b}$ that we assume to be bounded. This polytope has a set of vertices (i.e., basic feasible solutions) $\mathcal{V} = \{\boldsymbol{v} : A\boldsymbol{v} = \boldsymbol{b} \text{ and } v \text{ is basic feasible}\}$. It is known that any solution $\boldsymbol{x}$ of $A\boldsymbol{x} = \boldsymbol{b}$ can be expressed as a convex combination of these $\boldsymbol{v}$; that is, there exist positive scalars $\lambda_{\boldsymbol{v}}$ satisfying $\boldsymbol{x} = \sum_{\boldsymbol{v}\in\mathcal{V}} \lambda_{\boldsymbol{v}}\boldsymbol{v}$ such that $\lambda_{\boldsymbol{v}} \geq 0$ and $\sum_{\boldsymbol{v}\in\mathcal{V}} \lambda_{\boldsymbol{v}} = 1$. For any linear cost vector $\boldsymbol{c}$, we can also define a new linear cost vector $\boldsymbol{c}' = (\boldsymbol{c}^T\boldsymbol{v})_{\boldsymbol{v}\in\mathcal{V}}$ to be the cost vector in terms of the cost contribu-

tion from each vertex. An LP $\min_{\boldsymbol{x}} \left\{ \boldsymbol{c}^T \boldsymbol{x} : A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq 0 \right\}$ can then be expressed as $\min_{\boldsymbol{\lambda}} \left\{ \boldsymbol{c'}^T \boldsymbol{\lambda} : \sum_{\boldsymbol{v} \in \mathcal{V}} \lambda_{\boldsymbol{v}} = 1, \boldsymbol{\lambda} \geq 0 \right\}$.

This idea of re-expressing an LP in terms of its basic feasible solutions can be used as a decomposition method. Consider a matrix $A$ which can be expressed as two submatrices $A = \begin{pmatrix} A^L \\ A^D \end{pmatrix}$, where $A^L$ is called the *linking constraint matrix* and has no restrictions (as a double-meaning, $L$ can refer to the word "linking" and to the number of rows in this matrix), and $A^D$ is strictly block-diagonal, i.e., can be expressed as the direct sum (corner-to-corner composition) of submatrices $A_i^D$ for $i = 1, ..., I$ where $I$ is the number of blocks. Along similar lines, the RHS $b$ can be partitioned into $\begin{pmatrix} b^L \\ b^D \end{pmatrix}$. Following this, the variables $\boldsymbol{x}$, block right-hand-side $\boldsymbol{b}^D$, and cost vector $\boldsymbol{c}$ can be partitioned into $\boldsymbol{x}_i$, $\boldsymbol{b}_i^D$ and $\boldsymbol{c}_i$ to correspond to their respective blocks, so we can write $A_i^D \boldsymbol{x}_i = \boldsymbol{b}_i^D$ for all $i$. Ignoring the $A^L$ constraints for now, any block solution $\boldsymbol{x}_i$ can be expressed as a convex combination of block vertices $\boldsymbol{v} \in \mathcal{V}_i$ such that $\boldsymbol{x}_i = \sum_{\boldsymbol{v} \in \mathcal{V}_i} \lambda_{i,\boldsymbol{v}} \boldsymbol{v}$, and in turn any solution $\boldsymbol{x} = (\boldsymbol{x}_i)_{i=1}^I$ to $A^D \boldsymbol{x} = \boldsymbol{b}^D$ can be expressed in terms of the block-wise $\boldsymbol{\lambda}$. To transform $\boldsymbol{c}$ to be in terms of $i$, we define $\boldsymbol{c'} = (\boldsymbol{c}_i^T \boldsymbol{v})_{i \in [I], \boldsymbol{v} \in \mathcal{V}_i}$. Lastly, we must transform $A^L$, and do so by defining $A'^L$ to be the matrix with entries $[A'^L]_{l,(i,\boldsymbol{v}_i)} = [A_i^L]_l \boldsymbol{v}_i$, where $A_i^L$ is the submatrix of $A^L$ corresponding to the $i^{\text{th}}$ block, so the entries of the transformed linking constraint matrix describe the contribution of a vertex of a block to some $l^{\text{th}}$ linking constraint. The original partially block-diagonal problem can therefore be expressed as the following *master problem:*

$$
\begin{aligned}
&\min_{\boldsymbol{\lambda}} \boldsymbol{c'}^T \boldsymbol{\lambda} \\
&\text{s.t. } A'^L \boldsymbol{\lambda} = \boldsymbol{b}^L, \\
&\qquad \sum_{\boldsymbol{v} \in \mathcal{V}_i} \lambda_{i,v} = 1 \text{ for all } i \in [I], \\
&\qquad \boldsymbol{\lambda} \geq 0,
\end{aligned}
$$

where $[I] := \{x \in \mathbb{Z} : 1 \leq x \leq I\}$. We then define a *reduced master problem* (RMP) to be the master problem with subsets $\overline{\mathcal{V}_i} \subset \mathcal{V}_i$ of the vertex sets, and therefore only a subset of the corresponding $\boldsymbol{\lambda}$ variables. The master problem can be solved by iteratively solving the reduced master problem, and extending the subset of vertices under consideration via a column generation method, which in turn is done by identifying entering variables $\boldsymbol{v}'_i \in \mathcal{V}_i \backslash \overline{\mathcal{V}_i}$ with negative reduced cost. If we let $\boldsymbol{\sigma}$ represent the vector of optimal dual variables for the constraint $A'^L \boldsymbol{\lambda} = \boldsymbol{b}^L$ for RMP with vertex subsets $\overline{\mathcal{V}_i}$ for each $i$, and for each block $i$ let $\beta_i$ represent the dual variable for the sum-to-one constraint corresponding to that block, then the problem of finding a new vertex for a given block can be expressed as $\min_{\boldsymbol{v}' \in \mathcal{V}_i \backslash \overline{\mathcal{V}_i}} \left\{ \boldsymbol{c}_i^T \boldsymbol{v}' - \boldsymbol{\sigma}^T \boldsymbol{v}' - \beta_i \right\}$. This vertex-finding problem can in turn be solved by transforming back to the original variables: $\min_{\boldsymbol{x}_i} \left\{ \boldsymbol{c}_i^T \boldsymbol{x}_i - \boldsymbol{\sigma}^T \boldsymbol{x}_i - \beta_i : A_i^D \boldsymbol{x}_i = \boldsymbol{b}_i^D \right\}$, and we call this the $i^{th}$ *pricing subproblem*, or $\text{SP}_i$. A basic feasible solution to this subproblem will, if the objective function is negative, correspond to a block vertex to be added to the reduced master problem which will improve its objective value. This process is then repeated to add more and more variables to the RMP until it can no longer be improved, at which point it offers a solution to the master problem. If no improving solution (i.e., with negative reduce cost) can be found to *any* of the subproblems, then the algorithm terminates as the optimal solution has been found. For clarity, we present this in algorithmic form:

1. Initialise: Identify initial subsets of vertices $\{\mathcal{V}_i^{(0)}\}_{i \in [I]}$ such that the RMP with these subsets is feasible. Set $n \leftarrow 0$.

2. Repeat: Solve RMP with vertex subsets $\{\mathcal{V}_i^{(n)}\}_{i \in [I]}$, obtaining dual variables $\boldsymbol{\sigma}^{(n)}$ for the linking constraints and $\boldsymbol{\beta}^{(n)}$ for the sum-to-one constraints. For each $i$, solve $\text{SP}_i$, and add the solution $\boldsymbol{v}'_i$ to the vertex set, i.e., $\mathcal{V}_i^{(n+1)} \leftarrow \mathcal{V}_i^{(n)} \cup \{\boldsymbol{v}'_i\}$, if the optimal subproblem objective value (i.e., reduced cost) is negative. Otherwise, $\mathcal{V}_i^{(n+1)} \leftarrow \mathcal{V}_i^{(n)}$. Regardless, $n \leftarrow n+1$. If *none* of the reduced costs were negative, then TERMINATE (the optimal solution has been found).

This concludes the discussion of Dantzig-Wolfe decomposition for bounded LPs. Section 2.1.4 now follows with a discussion of how to extend this to MILPs.

### 2.1.4 Dantzig-Wolfe Decomposition for MILPs

Dantzig-Wolfe decomposition can be extended to MILPs using either column generation as a heuristic, or via an exact method known as *branch-and-price*. We first focus on the latter, as first introduced by Barnhart et al. (1998). To be precise, we consider a decision space $\mathcal{X} = \bigtimes_{i=1}^{I} \mathcal{X}_i$ with $\mathcal{X}_i = \mathbb{Z}^{n_1^{(i)}} \times \mathbb{R}^{n_2^{(i)}}$ such that the decision variables $\boldsymbol{x}$ can be expressed as $\boldsymbol{x}^T = (\boldsymbol{x}_1^T, ..., \boldsymbol{x}_I^T)$, and consider problems of the form:

$$\begin{aligned} &\min_{\boldsymbol{x} \in \mathcal{X}} \boldsymbol{c}^T \boldsymbol{x} \\ &\text{s.t. } A^L \boldsymbol{x} = \boldsymbol{b}^L, \\ &\qquad A_i^D \boldsymbol{x}_i = \boldsymbol{b}_i^D, \text{ for all } i \in [I], \end{aligned}$$

so we have a block-diagonal structure in a MILP setting. We assume all blocks to be bounded. We call this base problem the *original problem*, or OP. Suppose we were to proceed in solving this via branch-and-bound, then at each node of the tree we would have the linear relaxation of OP equipped with some extra constraints defining the branching decisions. These linear relaxations are of course also block-diagonal, and can therefore be solved via Dantzig-Wolfe decomposition via a pricing subproblem. However, these relaxations can be made tighter by forcing the pricing subproblem to only generate columns that are integer feasible, so that block solutions to the reduced master problem at any node will be continuous convex combinations of integer-feasible block solutions. This will provide tighter bounds at each node than simply solving the linear relaxation normally. Altogether, branch-and-price is the process of solving a block-diagonal MILP via branch-and-bound, solving the node relaxations via Dantzig-Wolfe decomposition with pricing subproblems, but re-constraining those subproblems

to produce integer-feasible solutions. These integral subproblems are in turn solved by standard MILP methods (cutting planes, branch-and-bound etc) or by bespoke problem-specific methods.

The above method can also be relaxed to provide heuristic solutions. For this, we focus only on solving the linear relaxation of the block-diagonal MILP (i.e., the root node of the BB tree) via Dantzig-Wolfe decomposition, again forcing the pricing subproblem to only return integer-feasible solutions. When no relaxation-improving integer-feasible solutions can be generated, we can then attempt to find a good solution to the integer problem using the columns generated, i.e., forcing the $\boldsymbol{\lambda}$ variables in the RMP to be binary so that only one column can be selected per block of the original problem. Note that in general this does not guarantee even returning a feasible solution, but it is known to work well. Dantzig-Wolfe decomposition will, in one form or another, be used in Chapter 5 and Chapter 6 as the backbone of decomposition-based techniques for solving large structured MDPs and MDP Design problems. This concludes our review of deterministic optimisation, and Section 2.2 now follows with an introduction to MDPs.

## 2.2 Markov Decision Processes

Markov decision processes, or MDPs, are a framework for a class of problems often described as sequential decision making under uncertainty. They are a generalisation of Markov chains where, at any state, one can take one of a variety of actions to influence the transition probabilities, and to influence the immediate cost. One can think of a state $s$ as being a problem that, upon "solving" it with an action $a$, presents you with a random new problem $s'$. As such, one must balance the immediate cost of an action in a state with the costs of the problems that may be presented in the future if such an action is taken. As such, MDPs provide a flexible framework for sequential decision

making under uncertainty. Section 2.2.1 recalls some of the basics and important results of stochastic processes and Markov chains, most of which can be found in introductory textbooks such as Florescu (2014) (this level of detail is necessary for us due to our use of long-run average behaviour). Section 2.2.2 extends Markov chains to *Markov reward processes* (MRPs) by equipping them with a cost function, and discusses the various ways of analysing the long-run cost performance of these processes, and Section 2.2.3 then extends this further to the MDP by introducing action spaces for each state, and discusses what it means to optimise such a process. These latter two subsections provide a summary of all the necessary information that can be found in Puterman (2014).

### 2.2.1 Markov Chains

For completeness, we first define a stochastic process in all generality, following thorough introductory texts such as (Florescu, 2014, Part II), which in turn requires us to return to the basics of probability theory, starting with the *$\sigma$-algebra*. Given a set $\mathcal{C}$, a set $\Sigma$ of subsets of $\mathcal{C}$ is called a *$\sigma$-algebra* over $\mathcal{C}$ if three properties are satisfied:

1. $\mathcal{C} \in \Sigma$,
2. if $\mathcal{Y} \in \Sigma$, then $\mathcal{C} \backslash \mathcal{Y} \in \Sigma$ (i.e., $\Sigma$ is closed under complements),
3. if $\mathcal{Y}_i \in \Sigma$ for $i = 1, ..., I$ for some finite $I \in \mathbb{N}$, then $\bigcup_{i \in [I]} \mathcal{Y}_i \in \Sigma$, (i.e., $\Sigma$ is closed under finite unions).

Using this, we can define a *probability space* $(\Omega, \mathcal{F}, \mathbb{P})$ where $\Omega$ is a set called the *sample space*, $\mathcal{F}$ is a $\sigma$-algebra over $\Omega$ called the *event space*, and $\mathbb{P} : \mathcal{F} \to [0, 1]$ is a function called the *probability measure*, which must satisfy $\mathbb{P}(\Omega) = 1$, and for any finite collection $\mathcal{Y}_1, ..., \mathcal{Y}_I$ of disjoint elements of $\mathcal{F}$ we have $\mathbb{P}(\bigcup_{i \in [I]} \mathcal{Y}_i) = \sum_{i=1}^{I} \mathbb{P}(\mathcal{Y}_i)$. We define a *random variable* to be a function $X : \Omega \to \mathcal{S}$, and this random variable is called $\mathcal{F}$-measurable if for any subset $\mathcal{X} \subset \mathcal{S}$, the pre-image $X^{-1}(\mathcal{X})$ is in $\mathcal{F}$. In turn, the set $\sigma(X) = \{X^{-1}(\mathcal{X}) : \mathcal{X} \subset \mathcal{S}\}$ is called the $\sigma$-algebra generated by $X$. Given an index

(i.e., totally ordered) set $\mathcal{T}$, a *stochastic process* is then an indexed set of $\mathcal{F}$-measurable random variables $S = \{S_t : \Omega \to \mathcal{S}\}_{t\in\mathcal{T}}$. When $\mathcal{T}$ is continuous, we may also write $\{S(t)\}_{t\in\mathcal{T}}$, however note that each $S(t)$ is still a function of $\omega \in \Omega$ (i.e., it is a random variable, not a value). A probability space can be equipped with a *filtration*, which is a sequence $\mathbb{F} = \{\mathcal{F}_t\}_{t\in\mathcal{T}}$ of $\sigma$-algebras with the same index set, such that $\mathcal{F}_t \subset \mathcal{F}_{t^+}$ for all $t < t^+$. A filtration of use to us is the filtration of $\sigma$-algebras generated by the stochastic process $\{S_t\}_{t\in\mathcal{T}}$, given by $\mathcal{F}_t = \sigma\left(\{S_{t^-}\}_{t^-\leq t}\right)$, i.e., the $\sigma$-algebra generated by the partial stochastic process $\{S_{t^-}\}_{t^-\leq t}$ when viewed as a single random variable from $\Omega$ to $\mathcal{S}^{\mathcal{T}_{t^-}}$ where $\mathcal{T}_{t^-} = \{t' \in \mathcal{T} : t' \leq t^-\}$, or as a random function from $\Omega \times \mathcal{T}_{t^-}$ to $\mathcal{S}$ (these are equivalent). These filtrations generated by stochastic processes up to an index $t$ can be interpreted as encoding all possible histories or trajectories of the process up until that time.

All of this theoretical background is required to define the *Markov property* in full generality. Following Durrett (2019), we say that a stochastic process $\{S_t\}_{t\in\mathcal{T}}$ has the Markov property if:

$$\mathbb{P}(S_{t'} \in \mathcal{X} \,|\, \mathcal{F}_t) = \mathbb{P}(S_{t'} \in \mathcal{X} \,|\, S_t), \text{ for all } \mathcal{X} \subset \mathcal{S}, t < t'.$$

That is, at a time $t'$, given the history of the process up until a time $t < t'$, the probability of $S_{t'}$ taking on any value or belonging to any subset of values depends only on the most recent (highest indexed) value taken by the process $(S_t)$ in that history, and not the trajectory that the process took to obtain that value. This is commonly known as *memorylessness*. Any stochastic process that satisfies the Markov property is known as a *Markov process*. We will purely focus on the case where $\mathcal{S}$, the set of values attainable by the process, is finite. There are then two alternatives we consider for the index set: $\mathcal{T} = \mathbb{Z}_{\geq 0}$, which results in a *discrete-time* Markov chain (DTMC); and $\mathcal{T} = \mathbb{R}_{\geq 0}$, which results in a *continuous-time* Markov chain.

We begin with the discrete-time case, where any such chain can be fully specified

by an initial distribution $\boldsymbol{\pi}_0 \in [0,1]^{|\mathcal{S}|}$ such that $\boldsymbol{\pi}_0(s) := \mathbb{P}(S_0 = s)$ for all $s$, and the *transition probabilities* $p(s,s') := \mathbb{P}(S_{t+1} = s' \mid S_t = s)$ for any $t \in \mathbb{Z}_{\geq 0}$. These can be expressed as a (one-step) transition rate matrix $P$ where $[P]_{s,s'} = p(s,s')$. We can take this further by considering the $k$-step transition matrix, which is known to be $P^{(k)} = P^k$, and write $p^{(k)}(s,s')$ as the probability of moving from state $s$ to $s'$ in exactly $k$ steps. An object of interest is $\boldsymbol{\pi}_t := (\pi_t(s))_{s\in\mathcal{S}}$ where $\boldsymbol{\pi}_t(s) = \mathbb{P}(S_t = s)$, so that $\boldsymbol{\pi}_t$ is a vector of probabilities of being in a particular state $s$ at time $t$. Particularly of interest is whether or not the expression "$\boldsymbol{\pi}_\infty := \lim_{t\to\infty} \boldsymbol{\pi}_t$" (the limiting distribution of the chain) makes sense, and what this would look like. The first step towards this is the concept of a *stationary distribution*, which is a probability vector $\boldsymbol{\pi}$ satisfying $\boldsymbol{\pi}^T = \boldsymbol{\pi}^T P$ (note that in general this need not be unique). It intuitively makes sense that $\boldsymbol{\pi}_\infty$ should satisfy this expression if it exists; however, we need additional assumptions to guarantee that this value is unique, and that the convergence actually happens. For this, we require further theoretical development.

We say a state $s$ *communicates* with $s'$ if and only if $P^{(k)}_{s,s'}, P^{(k')}_{s',s} > 0$ for some $k, k' < \infty$, and write $s \leftrightarrow s'$. In short, this means $s'$ is "reachable" from $s$ with non-zero probability in a finite number of steps, and vice versa. This is known to be an equivalence relation on $\mathcal{S}$, partitioning the space into sets called *communicating classes*. A communicating class $\widehat{s}$ is called *closed* if for any $s \in \widehat{s}$, $s' \notin \widehat{s}$, $p(s,s') = 0$, meaning once the chain enters the class, it cannot escape. For finite state Markov chains, this is closely related to the concept of (positive) recurrence, meaning that the expected number of transitions from leaving a state to returning to that same state is finite. A state without this property is called transient. Both recurrence and transience are *class properties*, meaning that if they are true for one state in a class, they are true for all states in a class. A Markov chain with only one recurrent class is called unichain, and this is the necessary condition for the existence of a *unique* stationary distribution $\boldsymbol{\pi}$. To ensure that $\boldsymbol{\pi}_t \to \boldsymbol{\pi}$ as $t \to \infty$, we need the additional assumption

of *aperiodicity*. We say that a state $s$ has *period* $k$ if $k$ is the highest common factor (aka greatest common divisor) of the set $\{t > 1 : p^{(t)}(s,s) > 0\}$ (the set of numbers of time-steps such that $s$ is reachable from itself in that many steps), and is *aperiodic* if $k = 1$. Additionally, aperiodicity is a class property. So, if a Markov chain has a single recurrent class and this class is aperiodic, then a unique stationary distribution $\boldsymbol{\pi}$ exists and we have $\boldsymbol{\pi}_t \to \boldsymbol{\pi}$ as $t \to \infty$ for any initial distribution $\boldsymbol{\pi}_0$.

We must also introduce the *continuous-time* Markov chain (CTMC), i.e., the case where $\mathcal{T} = \mathbb{R}_{\geq 0}$, which is fully specified again by an initial distribution $\boldsymbol{\pi}_0(s) := \mathbb{P}(S(0) = s)$, and *transition rates* $q(s,s')$ which, for $s \neq s'$, can be interpreted as the parameter of an Exponential distributed random variable giving the time until a transition from $s$ to $s'$. These must always satisfy $q(s,s) = -\sum_{s' \neq s} q(s,s')$. We begin with some loose intuition via some abuse of notation, and begin by imagining these Exponential transition times for each $s'$ "competing" to be the transition that actually happens. Given that $S(t) = s$, we can define $E_{s,s'}(t) \sim \text{Exp}(q(s,s'))$ to be the transition times for each possible $s'$, $E_s(t) = \min_{s'} \{E_{s,s'}(t)\}$ to be the fastest transition time, and then our next state will be $S(t + E_s) = \arg\min_{s'} \{E_{s,s'}\}$, i.e., the state that achieves that fastest transition. For $t < t' < t + E_s(t)$, $S(t') = s$, i.e., the process remains in the same state until transition. To be more precise (and to evade any further abuse of notation for expository purposes), a continuous-time stochastic process $S = \{S(t) : \Omega \to \mathcal{S}\}_{t \in T}$ is a continuous-time Markov chain with transition rate matrix $Q$ ($[Q]_{s,s'} = q(s,s')$ for all $s, s'$) if $P(S(t+h) = s' \,|\, S(t) = s) = \delta_{i,j} + q(s,s')h + o(h)$, where $o(h)$ is any function such that $o(h)/h \to 0$ as $h \to 0$. This can be interpreted as the probability of transitioning from $s$ to $s' \neq s$ over a very small time duration $h$ as being roughly equal to $q(s,s')h$, and the probability of remaining in the same state being $1 - \sum_{s' \neq s} q(s,s')h$. Again, we are often interested in $\boldsymbol{\pi}_t = (\mathbb{P}(S(t) = s))_{s \in \mathcal{S}}$, and its limiting form $\boldsymbol{\pi}_\infty = \lim_{t \to \infty} \boldsymbol{\pi}_t$. The concept of a stationary distribution exists for the CTMC, being any probability vector $\boldsymbol{\pi}$ satisfying $\boldsymbol{\pi}^T Q = 0$. The same notions of

communicating classes, positive recurrence, and transience hold for CTMCs, however the notion of aperiodicity need not be considered due to the arbitrary and continuous waiting times of each state. A CTMC therefore has a unique stationary distribution if it is unichain (has one closed recurrent class), and $\boldsymbol{\pi}_t$ always converges to this distribution regardless of initial distribution.

This concludes all prerequisite knowledge for DTMCs and CTMCs, and we now follow with the introduction of Markov reward processes, or MRPs.

### 2.2.2 Markov Reward Processes

A Markov reward process (MRP) is simply a DTMC or CTMC equipped with a cost function $c : \mathcal{S} \to \mathbb{R}$ over the states of the chain (broadly speaking, we could allow for stochastic or time-heterogeneous cost functions $c_t$, but that is beyond the scope of this work). For DTMCs, $c$ is a lump-sum cost incurred every time the process transitioning into state $s$, whereas for CTMCs $c$ is a cost-rate, meaning an amount incurred per unit time spent in the state. Of interest to us are the ways in which we can define and evaluate the long-term performance of such a process. In the cost (bigger is worse) setting, these are generally termed the *cost-to-go*, whereas in the reward (bigger is better) setting, these are termed as *value functions*. The simplest class of cost-to-go functions are the *finite-horizon* cost-to-go functions:

$$J_T(s) = \mathbb{E}_S \left\{ \sum_{t=0}^{T} c(S_t) \,|\, S_0 = s \right\} \text{ (discrete time)},$$
$$J_T(s) = \mathbb{E}_S \left\{ \int_0^T c\left(S(t)\right) dt \,|\, S(0) = s \right\} \text{ (continuous time)},$$

where the expectations are taken with respect to the whole process $S$ (note that these finite-horizon costs-to-go are well-defined for any stochastic process equipped with a cost function, not just MRPs). These expressions provide the cumulative cost incurred up until a time (i.e., the *horizon)* $T$ when starting from a state $s$, and can be expressed

in recursive form:

$$J_T(s) = c(s) + \mathbb{E}_{S' \sim p(s,\cdot)} J_{T-1}(S') \text{ (discrete time)}$$
$$J_T(s) = c(s)t(s) + \mathbb{E}_{S' \sim q(s,\cdot)} J_{T-t(s)}(S') \text{ (continuous time)},$$

where in the discrete time case, $S' \sim p(s,\cdot)$ is a random variable (not a process!) over $\mathcal{S}$ with probability mass function $p(s,\cdot)$, representing the next state after transition; and in the continuous time case, $t(s) = -1/q(s,s)$ is the expected sojourn time from $s$ and $S' \sim q(s,\cdot)$ is a random variable over $\mathcal{S}$ with $\mathbb{P}(S' = s) = 0$ and remaining probabilities $\mathbb{P}(S' = s') \propto q(s,s')$, similarly representing the next state after transition. Note that this is only valid for $T > t(s)$.

The next type of cost-to-go function is the *infinite-horizon discounted* type, meaning the whole future is taken into account, subject to exponential decay with a *discount factor* $\gamma$. These cost-to-go functions are defined as follows:

$$J_\gamma(s) := \mathbb{E}_S \left\{ \sum_{t=0}^{\infty} \gamma^t c(S_t) \,|\, S_0 = s \right\} = c(s) + \gamma \mathbb{E}_{S' \sim p(s,\cdot)} J_\gamma(S') \text{ (discrete time)},$$
$$J_\gamma(s) := \mathbb{E}_S \left\{ \int_0^{\infty} \gamma^t c\left(S(t)\right) dt \,|\, S(0) = s \right\} = \frac{\gamma^{t(s)} - 1}{\ln \gamma} c(s) + \gamma^{t(s)} \mathbb{E}_{S' \sim q(s,s')} J_\gamma(S')$$
(continuous time).

Due to the geometric decay when $0 \leq \gamma < 1$, these series always converge so the cost-to-go functions are always well defined. The discounted infinite-horizon case is the one most often used in the MDP literature due to being a valid infinite horizon objective whilst not needing to worry about properties such as unichain or aperiodicity. However, the next type of cost-to-go function, and the one of interest for the remainder of our work, does rely on such properties.

The final long-run measure of performance focuses on the limiting cost behaviour of the reward process. We assume aperiodicity throughout. We can define the *long-run*

*average cost*, otherwise known as the *gain*, from $s$ in terms of the limit of the finite-horizon cost-to-go: $g(s) = \lim_{T\to\infty} \frac{J_T(s)}{T}$. This definition does hold for periodic chains when the limit is replaced with the Cesaro limit, but that is beyond the scope of this work. In the case of unichain Markov chains, the function $g(s)$ becomes a constant $g := \mathbb{E}_{S\sim\boldsymbol{\pi}}\{c(S)\}$, where $\boldsymbol{\pi}$ is the unique stationary distribution. While $g$ provides a measure for long-run performance of the chain as a whole (or, in the non-unichain class setting, a measure for each class), it does not provide a measure of performance for each state. For this, we define the *bias* as follows:

$$h(s) := \mathbb{E}_S\left\{\sum_{t=0}^{\infty}(c(S_t) - g(S_t)) \mid S_0 = s\right\} \text{ (discrete time)},$$
$$h(s) := \mathbb{E}_S\left\{\int_0^{\infty}(c(S(t)) - g(S(t)))\,dt \mid S(0) = s\right\} \text{ (continuous time)}.$$

This can be interpreted as the expected difference between the cumulative cost and the cumulative gain. For $s, s'$ in the same recurrent class, we also have $h(s) - h(s') = \lim_{T\to\infty}\{J_T(s) - J_T(s')\}$, so that the difference between the $h$ values gives the long-run total difference in cumulative cost between starting in state $s$ and state $s'$. One can use this to define a *relative cost-to-go* function for any state $s$ in the same recurrent class as some *reference state* $s_0$, and we define this by $h(s \mid s_0) := \lim_{T\to\infty}\{J_T(s) - J_T(s_0)\}$. In the unichain case where $g(s)$ is a constant $g$, we can express the bias as $h(s) = \lim_{T\to\infty}\{J_T(s) - gT\}$. As with the previous cases, $h$ can be expressed in recursive form as follows:

$$h(s) = (c(s) - g(s)) + \mathbb{E}_{S'\sim p(s,\cdot)}h(S') \text{ (discrete time)},$$
$$h(s) = (c(s) - g(s))\,t(s) + \mathbb{E}_{S'\sim q(s,\cdot)}h(S') \text{ (continuous time)}.$$

We now follow with Section 2.2.3, where we extend the MRP under long-run average costs to a decision making framework known as a Markov decision process.

### 2.2.3 Markov Decision Processes

Markov decision processes, or MDPs, are an extension of MRPs to include *actions* that influence the transitions and the costs. We equip the MRP with a collection of *feasible action spaces* $\{\mathcal{A}(s)\}_{s\in\mathcal{S}}$, and define the global action space as $\mathcal{A} := \bigcup_{s\in\mathcal{S}} \mathcal{A}(s)$. We denote the set of feasible state-action pairs by $\mathcal{SA} := \{(s,a) : s \in \mathcal{S}, a \in \mathcal{A}(s)\}$. In discrete time, the transition probabilities $p(s,s')$ are replaced by $p(s,a,s')$ for $a \in \mathcal{A}(s)$; likewise, in continuous time the transition rates are replaced by $q(s,a,s')$. The cost functions $c(s)$ are replaced by $c(s,a)$. A discrete-time MDP is defined as a tuple:

$$\mathfrak{D} := \langle \mathcal{S}, (\mathcal{A}(s))_{s\in\mathcal{S}}, p : \mathcal{SA} \times \mathcal{S} \to [0,1], c : \mathcal{SA} \to \mathbb{R} \rangle,$$

where $p$ is the transition probability function and $c$ is the one-step cost function. Similarly, a continuous-time MDP (CTMDP) is defined as a tuple:

$$\mathfrak{C} = \langle \mathcal{S}, (\mathcal{A}(s))_{s\in\mathcal{S}}, q : \mathcal{SA} \times \mathcal{S} \to \mathbb{R}, c : \mathcal{SA} \to \mathbb{R} \rangle,$$

where $q$ is the transition rate function. In either case, our goal is to somehow "control" our MDP to yield an MRP with minimal long-run average cost (finite-horizon cost and discounted infinite-horizon costs are of course common, but not relevant to our work).

An all-inclusive exploration of the many ways to potentially control an MDP can generally be avoided in the single-objective case due to famous results on the simplifying properties of optimal "policies" (we will define these shortly). However, these notions do not automatically hold true for multi-objective MDPs, and we therefore provide an unabridged understanding of control for MDPs. Before this, we must define a *set of histories* to be: the set of $t$-transition trajectories:

$$\mathcal{H}_t = \left\{ (s_0, a_0, s_1, a_1, ..., s_t) \in (\mathcal{S} \times \mathcal{A})^t : a_{t'} \in \mathcal{A}(s_{t'}) \text{ for all } 0 \leq t' < t \right\}$$

with the convention $\mathcal{H}_0 = \mathcal{S}$, for discrete time; and the set of $k$-transition trajectories:

$$\mathcal{H}_k = \{(0, s_0, a_0, t_1, s_1, a_1, ..., t_k, s_k) \in (\mathbb{R}_{\geq 0} \times \mathcal{S} \times \mathcal{A}) : a_{k'} \in \mathcal{A}(s_{k'}) \text{ for all } 0 \leq k' < k\},$$

for continuous time, again with convention $\mathcal{H}_0 = \mathcal{S}$. Note that the continuous-time histories do not allow for the action applied to be changed during the time spent in a state. The $t_{k'}$ values on the continuous-time histories represent the sojourn times from the previous state $s_{k'}$ under action $a_{k'}$. While cumbersome, these notions of histories are necessary because, in all generality, one could in principle make decisions (i.e., apply an action to a state at a given point in time) based on the whole history of the process so far. The basic units of decision making in MDPs are known as the *decision rules*, of which there are four main types:

- a *Markovian deterministic* decision rule is a function $d : \mathcal{S} \to \mathcal{A}$ such that $(s, d(s)) \in \mathcal{SA}$ for all $s$,
- a *history-dependent deterministic* decision rule is a function $d_t : \mathcal{H}_t \to \mathcal{A}$ such that $(s_t, d(h_t)) \in \mathcal{SA}$ for all $(s_0, a_0, ..., s_t) \in \mathcal{H}_t$,
- a *Markovian randomised* decision rule is a function $d : \mathcal{S} \times \mathcal{A} \to [0, 1]$ such that $d(s, \cdot)$ is a probability mass function over $\mathcal{A}(s)$ for all $s$,
- a *history-dependent randomised* decision rule is a function $d_t : \mathcal{H}_t \times \mathcal{A} \to [0, 1]$ such that for all $h_t = (s_0, a_0, ..., s_t) \in \mathcal{H}_t$, $d_t(h_t, \cdot)$ is a probability mass function over $\mathcal{A}(s_t)$.

Of course, we edit the history-dependent definitions in the continuous-time case to adjust for the inclusion of sojourn times in the histories. A decision rule tells us what action to take for one transition of the decision process. To control the whole process, we use a *policy*, which is a sequence of decision rules $\{d_t\}_{t \in \mathbb{Z}_{\geq 0}}$. A policy that satisfies $d_t = \mu$ for all $t$ and some Markovian decision rule $\mu$ is called a *stationary* policy, and

commonly (when this is not confusing) we call the decision rule $\mu$ itself the stationary policy. An important subclass of stationary policies are *deterministic stationary* (DS) polices, where the decision rule $\mu$ is deterministic. An MDP or CTMDP equipped with a DS policy $\mu$ always induces an MRP with transition probabilities given by $p_\mu(s, s') := p(s, \mu(s), s')$ (or transition rates given by $q_\mu(s, s') := q(s, \mu(s), s')$) and one-step costs (or cost rates) given by $c_\mu(s) := c(s, \mu(s))$. More generally, an MDP equipped with any policy induces a stochastic process with costs, but the details of this are beyond the scope of this work. DS policies not only allow us to control the process in a simple and intuitive way, but also allow us to classify processes based on their transitional structure with respect to DS policies. Some important classifications are as follows:

- An MDP is *unichain* if the MRP induced by any DS policy $\mu$ is also unichain;
- An MDP is *weakly communicating* if there exists a closed set of states $\mathcal{S}^r \subset \mathcal{S}$ such that for all pairs of states $s, s' \in \mathcal{S}^r$ there exists a DS policy $\mu$ such that $s'$ is *accessible* from $s$ (i.e., the probability of being in state $s'$ at some point in time when starting from $s$ is non-zero), and for all $s \in \mathcal{S} \backslash \mathcal{S}^r$, $s$ is transient under all DS policies.

It is known that unichain MDPs are a subclass of weakly communicating MDPs. For any weakly communicating single-objective MDP, it is well-known that there exists a deterministic stationary *optimal* policy with only one recurrent class (Puterman, 2014, Theorem 8.3.2). As such, if we take $\mathcal{M}$ to be the set of deterministic stationary policies that induce a unichain MRP, the problem of optimising an MDP can be represented as:

$$\min_{\mu \in \mathcal{M}} g(\mu),$$

where $g(\mu)$ is the long-run average cost of the MRP induced by $\mu$. Discrete-time MDPs can be optimised in a variety of ways, including value iteration, policy iteration, and

linear programming; however, we do not optimise any discrete-time MDPs in this work (discrete-time MDPs are only used for some proofs), so we omit these methods. Instead, we provide a linear programming formulation for CTMDPs under the long-run average cost criterion:

$$\begin{aligned}
\min_{\boldsymbol{\pi}} \quad & \sum_{(s,a)\in\mathcal{SA}} c(s,a)\pi(s,a) \\
\text{s.t.} \quad & \sum_{(s,a)\in\mathcal{SA}} q(s,a,s')\pi(s,a) = 0, \text{ for all } s' \in \mathcal{S} \\
& \sum_{(s,a)\in\mathcal{SA}} \pi(s,a) = 1, \\
& \pi(s,a) \geq 0 \text{ for all } (s,a) \in \mathcal{SA}.
\end{aligned}$$

The decision variables $\pi(s,a)$ are known as *state-action frequencies*, and represent the long-run proportion of time spent in state $s$ taking action $a$. The constraints ensure that $\boldsymbol{\pi}$ is a valid probability distribution satisfying the CTMDP equivalent of the balance equations $\boldsymbol{\pi}^T Q = 0$ from Section 2.2.1. Each feasible solution $\boldsymbol{\pi}$ corresponds to a randomised stationary policy $\mu_\pi(s,a) = \frac{\pi(s,a)}{\sum_{a'\in\mathcal{A}(s)}\pi(s,a')}$, and each basic feasible solution satisfies $\pi(s,a) > 0$ for at most one value of $a$ (for transient states, $\pi(s,a) = 0$ for all actions), and corresponds to a DS policy $\mu_{\boldsymbol{\pi}}(s) = \arg\max_{a\in\mathcal{A}(s)} \pi(s,a)$ for all recurrent $s$. In the other direction, a policy $\mu$ that induces a unichain MRP with stationary distribution $\boldsymbol{\pi}_\mu$ corresponds to a basic feasible solution

$$\pi(s,a) = \begin{cases} \pi_\mu(s), & a = \mu(s), \\ 0, & \text{otherwise.} \end{cases}$$

For a weakly communicating MDP, it is known that there always exists a unichain optimal policy, and that an optimal basic feasible solution to the LP will always correspond to such a policy. This LP formulation will form the basis of how we solve MDPs of a

reasonable size, and serves as the basis for the construction of MDP Design problems in Chapter 3, Chapter 4, and Chapter 6. To conclude our introduction of MDPs, we discuss the equivalence between discrete-time and continuous-time MDPs that will be useful in some of the proofs in our work.

### 2.2.4 Uniformisation

*Uniformisation* is a method introduced by Lippman (1975), and expanded upon by Serfozo (1979), that converts a continuous-time MDP under the discounted infinite-horizon or long-run average cost criterion (we focus on the latter) into a discrete-time MDP. This is useful both to make CTMDPs compatible with discrete-time solution methodologies (value iteration, policy evaluation, reinforcement learning, etc), and for the purposes of constructing proofs. Given a CTMDP $\mathfrak{C} = \langle \mathcal{S}, \mathcal{A}, q, c \rangle$, we choose a *uniformisation parameter* satisfying $0 < \Delta \leq -1/\max_{(s,a)\in\mathcal{SA}} q(s,a,s)$, and define a discrete-time MDP $\mathfrak{D}_\Delta = \langle \mathcal{S}, \mathcal{A}, p_\Delta, c_\Delta \rangle$ with the same state space and feasible action spaces, and transition probabilities

$$p_\Delta(s,a,s') = \begin{cases} q(s,a,s')\Delta, & s \neq s', \\ 1 + q(s,a,s)\Delta, & s = s', \end{cases}$$

and one-step costs $c_\Delta(s,a) = c(s,a)\Delta$. The decision processes $\mathfrak{C}$ and $\mathfrak{D}_\Delta$ are *equivalent* in the sense that, for any DS policy, the MRP induced by the policy will have the same stationary distribution(s) and therefore the same long-run average cost (they will also be equivalent in discounted cost-to-go, but this is not relevant for our work). Notably, an optimal DS policy for one is also optimal for the other. This concludes the prerequisite knowledge for MDPs, and Section 2.3 now follows with an introduction to bi-objective optimisation.

## 2.3 Bi-Objective Optimisation

Bi-objective optimisation is a concept that extends mathematical optimisation from the minimisation of one objective function to the optimal balancing of two objective functions in a (possibly infinite) number of ways. To understand the implications of this, in Section 2.3.1 we first introduce the concept of Pareto optimality, which extends the concept of optimality to the multi-objective case. Bi-objective problems are generally solved via *scalarisation*, which is a process by which a bi-objective problem (the problem of finding all or some Pareto optimal solutions) is transformed into a set of single-objective problems that can more readily be solved. In Section 2.3.2 we introduce our first scalarisation method, known as the mixed-objective method, and in Section 2.3.3 we introduce the other scalarisation method, known as the $\varepsilon$-constraint method. We then explore the implications of bi-objective optimisation for MDPs in Section 2.3.4.

### 2.3.1 Pareto Optimality

While our work focuses on bi-objective optimisation, we provide the notion of Pareto-optimality for the general multi-objective case. In single-objective optimisation, the notion of optimality is quite intuitive: given a single-objective deterministic optimisation problem $\min_{x\in\mathcal{X}} f(x)$, a solution $x^*$ is optimal if $f(x^*) = \min_{x\in\mathcal{X}} f(x)$. It is fruitful to consider an alternative definition: a solution $x$ is optimal if there does not exist a different solution $x' \neq x$ such that $f(x') < f(x)$. If this were the case, then we might say that $x'$ *dominates* $x$, and that a solution $x$ is optimal if and only if there does not exist a solution $x'$ that dominates $x$. We can extend this notion of domination to the multi-objective case.

**Definition 2.3.1** (Domination)**.** *Given a vector-valued function* $\boldsymbol{f} : \mathcal{X} \to \mathbb{R}^K$ *with* $\boldsymbol{f}(x) = (f_1(x), ..., f_K(x))$*, and two elements* $x, x' \in \mathcal{X}$*, we say* $x'$ $\boldsymbol{f}$-dominates $x$ *if* $f_k(x') \leq f_k(x)$ *for all* $k \in [K]$ *and* $f_k(x') < f_k(x)$ *for at least one* $k \in [K]$*.*

When the context is clear, we simply say $x'$ dominates $x$. Of course, the inequalities are flipped if we are in the maximisation setting instead of the minimisation setting (as we will be for the forthcoming examples in this section). We can then say that a solution $x \in \mathcal{X}$ is *Pareto-optimal* if there does not exist a solution $x'$ that dominates $x$. Such a solution can be thought of as a unique and optimal balancing of the multiple objectives, such that it is impossible to improve across any objective without sacrificing in another. The set of all Pareto-optimal solutions in a set $\mathcal{X}$ with respect to a vector function $\boldsymbol{f}$ is often called the *Pareto front*, denoted by $\mathrm{PF}_{\boldsymbol{f}}(\mathcal{X})$, however this term can also be used to refer to the set $\mathrm{OPF}_{\boldsymbol{f}}(\mathcal{X}) = \{\boldsymbol{f}(x) : x \in \mathrm{PF}_{\boldsymbol{f}}(\mathcal{X})\}$ of function evaluations of Pareto-optimal solutions. For clarity, we refer to the latter as the objective Pareto front, and the former as the Pareto front.

Where the notion of optimising or solving a single-objective program is given by finding the optimal objective value and a corresponding optimal solution, the notion of optimising or solving a multi-objective program is given by finding the objective Pareto front and a corresponding set of solutions. In general, the number of Pareto optimal solutions can be infinite, and uncountably so in the case of a decision space with a continuous component as in mixed-integer linear programming. We can demonstrate this with a simple example.

**Example 2.3.1.** *Consider the following bi-objective linear program:*

$$\begin{aligned} &\max_{x,y} x \\ &\max_{x,y} y \\ &s.t.\, x + y \leq 2 \\ &x, y \geq 0 \\ &x, y \in \mathbb{R}. \end{aligned}$$

*Note first that we have chosen our objectives such that the decision space and the objec-*

*tive space are the same. This of course need not be the case generally, we just do this to produce easily visualised examples. It is clear that all solutions of the form* $(t, 2-t)$ *for* $t \in [0, 2]$ *are Pareto-optimal, and therefore the Pareto front is uncountably infinite.*

While this may seem daunting in general, it is known in the bounded LP case that there will only ever be finitely many Pareto-optimal basic feasible solutions, and if there are uncountably many non-basic feasible solutions, they will always simply be linear interpolations between two of the Pareto-optimal basic feasible solutions (or, in the general multi-objective case, linear combinations of Pareto-optimal basic feasible solutions, as per Ehrgott (2005)). We may now introduce two methods for scalarising bi-objective MILPs, and then investigate bi-objective MDPs.

### 2.3.2 Mixed-Objectives Method

The mixed-objectives method is a form of *scalarisation*, meaning it converts a bi-objective program into a single-objective problem for finding one of the Pareto-optimal solutions. For a bi-objective linear program $\min_{\boldsymbol{x}\in\mathcal{X}}(\boldsymbol{c}^T\boldsymbol{x}, \boldsymbol{d}^T\boldsymbol{x})$, performing the mixed-objectives method simply means choosing a *weight* $w \in [0, 1]$, defining a scalarised objective function $f_w(\boldsymbol{x}) = w\boldsymbol{c}^T\boldsymbol{x} + (1-w)\boldsymbol{d}^T\boldsymbol{x}$ and solving the single-objective problem $\min_{\boldsymbol{x}\in\mathcal{X}}\{f_w(\boldsymbol{x})\}$. Solving this problem for any $w$ will always return a basic Pareto-optimal solution. In the continuous LP case, where the objective Pareto front is piecewise-linear and convex (Ehrgott, 2005), solving for suitable values of $w$ using a simplex-based LP solver will yield basic feasible solutions corresponding to all of the "corners" of the objective Pareto front, with suitable linear interpolations providing the rest of the front. Note that this is *not* the same as finding all Pareto-optimal basic feasible solutions. Consider a three-dimensional polytope defined by linear inequalities, and suppose that for some value of $w$, a whole many-sided face of that polytope is optimal. However, for many vertices of that face, they may only be optimal for that one value of $w$, and for that one value, an LP solver will only return one of those vertices

as an optimal solution, meaning the others will be missed.

One method for searching all "interesting" values of $w$ is often referred to as the *dichotomic* method, introduced by Aneja and Nair (1979). The basic idea is that if we have two weights $w_1 < w_2$ and corresponding Pareto-optimal solutions $\boldsymbol{x}_{w_1}$ and $\boldsymbol{x}_{w_2}$, then we can choose $w_3$ such that $f_{w_3}(\boldsymbol{x}_{w_1}) = f_{w_3}(\boldsymbol{x}_{w_2})$. We then solve the mixed-objective problem with $w_3$ with the hope of finding a solution $\boldsymbol{x}_{w_3}$ such that $f_{w_3}(\boldsymbol{x}_{w_3}) < f_{w_3}(\boldsymbol{x}_{w_1})$. If this is successful, we repeat this recursively for $w_1, w_3$ and $w_3, w_2$ and so on to find more Pareto-optimal solutions. However, if $f_{w_3}(\boldsymbol{x}_{w_3}) = f_{w_3}(\boldsymbol{x}_{w_1})$, then we fail to find a new Pareto-optimal solution "in-between" $\boldsymbol{x}_{w_1}$ and $\boldsymbol{x}_{w_2}$. In the LP case where the $x_{w_i}$ are always basic feasible, this means the two solutions $\boldsymbol{x}_{w_1}$ and $\boldsymbol{x}_{w_2}$ are adjacent (i.e., connected by an edge of the feasible-region polytope). For completeness, we provide the general algorithm here:

- Initialize: Define $z_1^{(1)} = \min_{\boldsymbol{x}\in\mathcal{X}} \left\{\boldsymbol{c}^T\boldsymbol{x}\right\}$, $z_2^{(1)} = \min_{\boldsymbol{x}\in\mathcal{X}} \left\{\boldsymbol{d}^T\boldsymbol{x} \,|\, \boldsymbol{c}^T\boldsymbol{x} = z_1^{(1)}\right\}$ so that these are the Pareto-optimal objective values for the first objective. Likewise, define $z_2^{(2)} = \min_{\boldsymbol{x}\in\mathcal{X}} \left\{\boldsymbol{d}^T\boldsymbol{x}\right\}$, and $z_1^{(2)} = \min_{\boldsymbol{x}\in\mathcal{X}} \left\{\boldsymbol{c}^T\boldsymbol{x} \,|\, \boldsymbol{d}^T\boldsymbol{x} = z_2^{(2)}\right\}$. Define a queue $Q = [(1, 2)]$. Set $n = 2$.

- Repeat until $Q = \emptyset$: Pop an element $(r_1, r_2)$ off the queue $Q$. Set weights $w_1^{(r_1,r_2)} = |z_2^{(r_2)} - z_2^{(r_1)}|$ and $w_2^{(r_1,r_2)} = |z_1^{(r_2)} - z_1^{(r_1)}|$, then let $\overline{\boldsymbol{x}}$ be a solution of $\min_{\boldsymbol{x}\in\mathcal{X}} \left\{w_1^{(r_1,r_2)}\boldsymbol{c}^T\boldsymbol{x} + w_2^{(r_1,r_2)}\boldsymbol{d}^T\boldsymbol{x}\right\}$. Set $(\overline{z_1}, \overline{z_2}) = \left(\boldsymbol{c}^T\overline{\boldsymbol{x}}, \boldsymbol{d}^T\overline{\boldsymbol{x}}\right)$. If $(\overline{z_1}, \overline{z_2}) = (z_1^{(r_1)}, z_2^{(r_1)})$ (or equivalently by definition of $w$, $(\overline{z_1}, \overline{z_2}) = (z_1^{(r_2)}, z_2^{(r_2)})$), skip to the next iteration (we can't find a new solution). Otherwise, set $n \leftarrow n + 1$, record the solution $(z_1^{(n)}, z_2^{(n)}) = (\overline{z_1}, \overline{z_2})$, and add nodes $(r_1, n), (n, r_2)$ to the queue.

The setting of the weights $w_1^{(r_1,r_2)}$ and $w_2^{(r_1,r_2)}$ accomplishes the behaviour of defining a scalar objective function such that the solutions to problems $r_1$ and $r_2$ have equal objective value with this weighting. The method can be thought of as recursively searching for values of $w$ that return new Pareto-optimal solutions until no new solutions

can be found. When we apply this method in Chapter 4 and Chapter 6, we use our own slightly modified fork of MultiObjectiveAlgorithms.jl (Dowson et al., 2025) which can be found on GitHub.

Before moving onto an alternative method, we should first discuss the downsides of this method that could be addressed by such a method. The first downside is that the weights used in the mixed-objective method are not immediately interpretable, in the sense that given some weights, it is hard to predict any of the properties of the resulting Pareto optimal solution. Additionally, while the mixed-objectives method paired with linear interpolation can find the whole objective Pareto front, this is not true for MILPs. Arbitrary linear interpolations between integer-feasible points may not be integer themselves. Moreover, Pareto-optimal solutions may belong to the interior of the integer polytope, and will therefore not be optimal for any single linear objective. We demonstrate this by Example 2.3.2.

**Example 2.3.2.** *Consider the following bi-objective integer program:*

$$
\begin{aligned}
&\max_{x,y} x \\
&\max_{x,y} y \\
&\text{s.t. } \frac{3}{2}x + y \leq 3, \\
&\quad x \in \{0, 1, 2\}, \\
&\quad y \in \{0, 1, 2, 3\}.
\end{aligned}
$$

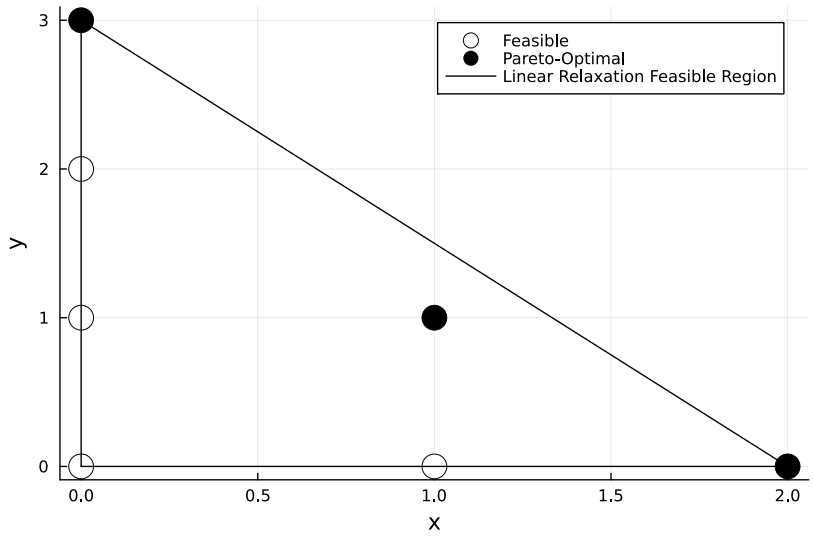



Figure 2.3.1: Visualisation of Example 2.3.2

*Note again that we have chosen our objectives such that the decision space and the*

*objective space are the same. The list of feasible solutions is:*

$$\{(0,0),(1,0),(2,0),(0,1),(1,1),(0,2),(0,3)\},$$

*and the Pareto front is given by* $PF = \{(2,0),(1,1),(0,3)\}$*. The set of all BFSs for the linear relaxation is given by* $\{(0,0),(2,0),(0,3)\}$*, so a MILP solver will always find an integral basic feasible solution for any mixed-objective weighting without any branching or cutting. However, the Pareto-optimal solution* $(1,1)$ *is not basic feasible, nor is it a linear interpolation between any two of the BFSs. As such, the mixed-objective method cannot find all Pareto-optimal solutions to this integer linear program.*

Using the terminology of Ehrgott (2005), the Pareto-optimal solutions that are optimal for a mixed-objective criterion are known as *supported* solutions, and are otherwise called non-supported (as with $(1,1)$ in Example 2.3.2). Section 2.3.3 now follows with an exploration of the $\varepsilon$-constraint method, which is capable of finding unsupported solutions.

### 2.3.3 Epsilon-Constraint Method

The $\varepsilon$-constraint method is the other main scalarisation method used for bi-objective optimisation. Rather than including both objectives in a unified objective function, this method moves one of the objectives into the constraints, and assigns a tunable parameter $\varepsilon$ to the right-hand-side of that constraint which acts as a "target" value for that objective. The other objective then becomes the sole objective of this augmented problem. To be concrete, if we have a bi-objective linear program $\min_{\boldsymbol{x}\in\mathcal{X}}(\boldsymbol{c}^T\boldsymbol{x}, \boldsymbol{d}^T\boldsymbol{x})$ and apply the method to the second objective, we obtain a problem $\min_{\boldsymbol{x}\in\mathcal{X}}\left\{\boldsymbol{c}^T\boldsymbol{x} \,|\, \boldsymbol{d}^T\boldsymbol{x} \leq \varepsilon\right\}$. In the case of continuous LPs, this will always return a Pareto-optimal solution if $z_2^{(2)} \leq \varepsilon \leq z_2^{(1)}$ where, as described in the dichotomic method, $z_2^{(1)}$ is the minimum value of the second objective that can be obtained while still optimally minimising the

first objective, and $z_2^{(2)}$ is the optimal value of the second objective alone. A value of $\varepsilon$ below this range will of course result in an infeasible problem, and a value of $\varepsilon$ above this range will result in a problem that successfully minimises the first objective, but does not guarantee Pareto-optimality if there exist multiple solutions that minimise the first objective but which differ in the second objective. The $\varepsilon$-constraint method offers an immediate benefit over the mixed-objectives method in that the role and effect of the $\varepsilon$ parameter is more clear than that of the weight $w$ in the mixed-objectives method. It can also more easily be tuned to achieve a desired "spacing" in the objective space of a partial (i.e., subset of the) Pareto front, which in practice can be very desirable. Similarly, if we know a range of desired objective values from expert knowledge, the $\varepsilon$-constraint method can easily specifically target that range.

As we alluded to in Section 2.3.2, the other benefit of the $\varepsilon$-constraint method is its ability to find unsupported solutions for MILPs, i.e., those solutions that cannot be found by the mixed-objectives method. To demonstrate this, we return to Example 2.3.2.

**Example 2.3.3.** *Consider the following $\varepsilon$-constrained version of Example 2.3.2:*

$$\begin{aligned} \max_{x,y}\ & y \\ \text{s.t. } & \frac{3}{2}x + y \leq 3, \\ & x \geq \frac{1}{2}, \\ & x \in \{0, 1, 2\}, \\ & y \in \{0, 1, 2, 3\}. \end{aligned}$$

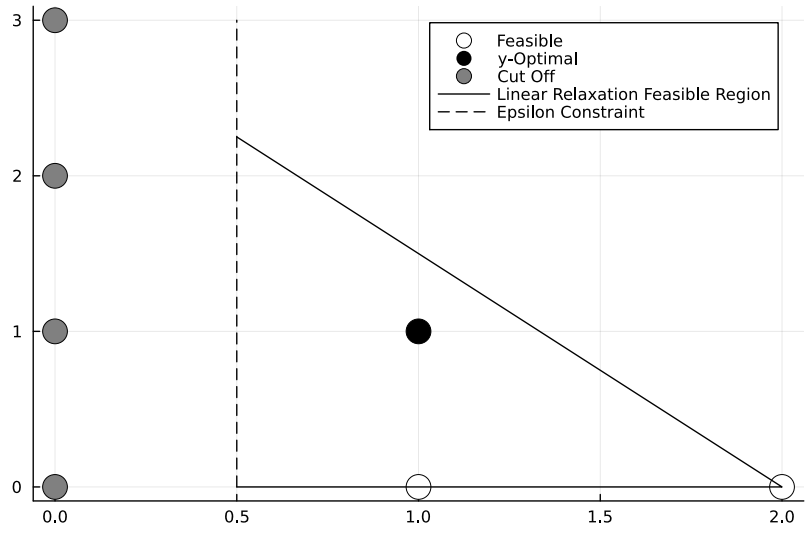



Figure 2.3.2: Visualisation of Example 2.3.3

*In Example 2.3.2, we saw that the mixed-objectives method only found the sup-*

*ported Pareto-optimal solutions* $\{(2,0),(0,3)\}$. *However, with* $\varepsilon$*-constraint* $x \geq 1/2$, *we "cut off" solutions* $\{(0,0),(0,1),(0,2),(0,3)\}$, *leaving us with feasible solutions* $\{(1,0),(1,1),(2,0)\}$. *Solution* $(1,1)$ *is returned as this maximises* $y$. *Therefore this method returns an unsupported Pareto-optimal solution.*

It is worth noting that the $\varepsilon$-constraint method may return Pareto-suboptimal solutions, which we shall also demonstrate by example.

**Example 2.3.4.** *Consider again the bi-objective problem from Example 2.3.3. Instead constrain the second objective and obtain the following problem:*

$$\begin{aligned} \max_{x,y}\ & x \\ \text{s.t. } & \frac{3}{2}x + y \leq 3, \\ & y \geq \frac{3}{2}, \\ & x \in \{0,1,2\}, \\ & y \in \{0,1,2,3\}. \end{aligned}$$

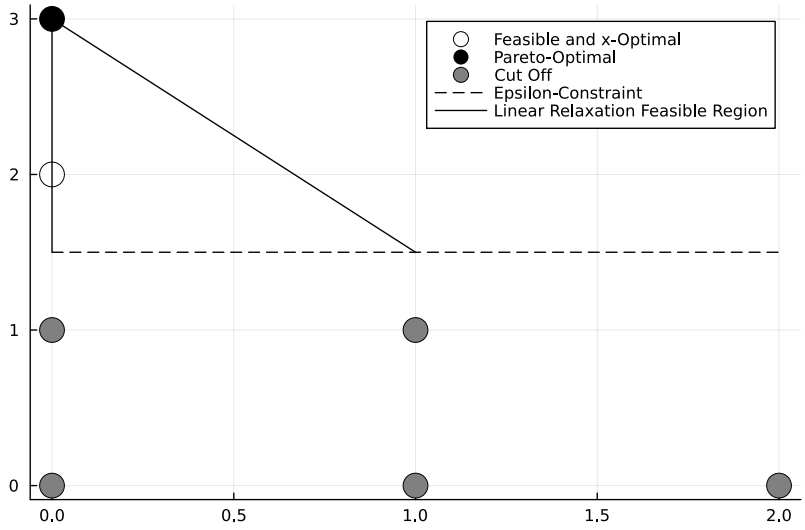



Figure 2.3.3: Visualisation of Example 2.3.4

*We now have feasible region* $\{(0,2),(0,3)\}$ *both of which have* $x = 0$, *so a solver may return either solution, even though* $(0,3)$ *clearly dominates* $(0,2)$ *(in the maximisation sense).*

To alleviate this, whenever one solves $z_1^\varepsilon = \min_{\boldsymbol{x}\in\mathcal{X}} \left\{\boldsymbol{c}^T\boldsymbol{x} \mid \boldsymbol{d}^T\boldsymbol{x} \leq \varepsilon\right\}$, one should also solve $z_2^\varepsilon = \min_{\boldsymbol{x}\in\mathcal{X}} \left\{\boldsymbol{d}^T\boldsymbol{x} : \boldsymbol{c}^T\boldsymbol{x} = z_1^\varepsilon\right\}$ to obtain a Pareto-optimal solution. However, with respect to the work carried out in Chapters 3-5, we seldom came across a case where the $\varepsilon$-constraint method returned a dominated solution, so this step was skipped.

The full $\varepsilon$-constraint algorithm for finding (most of) the Pareto front starts by optimising a single objective with no constraint, and repeatedly tightening the $\varepsilon$-constraint and reoptimising until the value of $\varepsilon$ would meet or pass the optimal objective value for the second objective, at which point the algorithm terminates. The tightening of the $\varepsilon$ value can be done using a parameter $\Delta\varepsilon > 0$, which can either be subtracted from the incumbent $\varepsilon$ for a linear decay, or multiplied by $\varepsilon$ (if $\Delta\varepsilon < 1$) for a geometric decay. To generalise this, suppose we have a decrement function $\mathrm{Dec} : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ such that $\mathrm{Dec}(\varepsilon) < \varepsilon$ for all $\varepsilon$. We provide the full algorithm below.

- Initialize: Define $z_1^{(1)} = \min_{\boldsymbol{x}\in\mathcal{X}} \left\{\boldsymbol{c}^T\boldsymbol{x}\right\}$, $z_2^{(1)} = \min_{\boldsymbol{x}\in\mathcal{X}} \left\{\boldsymbol{d}^T\boldsymbol{x} \,|\, \boldsymbol{c}^T\boldsymbol{x} = z_1^{(1)}\right\}$ so that these are the Pareto-optimal objective values for the first objective. Define $z_2^{(2)} = \min_{\boldsymbol{x}\in\mathcal{X}} \left\{\boldsymbol{d}^T\boldsymbol{x}\right\}$, $z_1^{(1)} = \min_{\boldsymbol{x}\in\mathcal{X}} \left\{\boldsymbol{c}^T\boldsymbol{x} : \boldsymbol{d}^T\boldsymbol{x} = z_2^{(2)}\right\}$. Set $\varepsilon = \mathrm{Dec}\left(z_2^{(1)}\right)$.
- Repeat until $\varepsilon < z_2^{(2)}$:
  - Set $z_1(\varepsilon) = \min_{\boldsymbol{x}\in\mathcal{X}} \left\{\boldsymbol{c}^T\boldsymbol{x} : \boldsymbol{d}^T\boldsymbol{x} \le \varepsilon\right\}$.
  - Set $z_2(\varepsilon) = \min_{\boldsymbol{x}\in\mathcal{X}} \left\{\boldsymbol{d}^T\boldsymbol{x} : \boldsymbol{c}^T\boldsymbol{x} = z_1(\varepsilon)\right\}$ with solution $x(\varepsilon)$.
  - Record the solution $x(\varepsilon)$ to the Pareto front and $(z_1(\varepsilon), z_2(\varepsilon))$ to the objective Pareto front.
  - Set $\varepsilon = \mathrm{Dec}\left(z_2(\varepsilon)\right)$.

Note that the decrement is *not* done in such a way as to avoid cutting off more than one Pareto-optimal solution, so this method may "leap-frog" solutions by decrementing by too much in one go, and therefore may not return the whole Pareto front. This contrasts with the dichotomic method which at least has the guarantee that all suitable values of $w$ will be probed. In general, the other major downside of using the $\varepsilon$-constraint method is that it affects the feasible region of the original problem, which may increase the difficulty for general methods such as exact solvers or metaheuristics, or render bespoke heuristics or solution methodologies unusable.

In our own work, we find the mixed-objectives method to be more suitable for some problems, namely MDPs due to their sensitive LP structure, and the $\varepsilon$-constraint method to be more suitable for others, especially integer problems. Section 2.3.4 now follows with a discussion of bi-objective MDPs.

### 2.3.4 Bi-Objective Optimisation for MDPs

A multi-objective (or vector) CTMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, q, \boldsymbol{c} \rangle$ where $\mathcal{S}, \mathcal{A}, q$ are the finite state space, finite action space, and transition rates as given before, and the cost-rate function $\boldsymbol{c} : \mathcal{SA} \to \mathbb{R}^K$, $\boldsymbol{c} = (c_1, ..., c_K)$ is now a vector-valued function. In turn, for any policy $\{d_t\}_{t \in \mathbb{Z}_{\geq 0}}$, we can define the vector finite-horizon cost-to-go for the induced stochastic process $\boldsymbol{J}_t(s) := (J_{k,t}(s))_{k=1}^K$, where $J_{k,t}$ is simply the scalar finite-horizon cost-to-go with respect to cost function $c_k$. In turn, we can define the vector long-run average cost $\boldsymbol{g}(s) = \lim_{t \to \infty} \boldsymbol{J}_t(s)/t$, whenever this limit exists (when it doesn't, one can take component-wise Cesaro limits, lim-sups, or lim-infs, but this is beyond the scope of this work). The goal is to find a set of policies (a Pareto front) that corresponds to the set of all non-dominated values of $\boldsymbol{g}$ (the objective Pareto front). Following White (1982), we know that there exist Pareto-optimal objective vectors that do not correspond to any deterministic stationary policy. From another perspective, for any multi-objective MDP, there may exist deterministic stationary policies that are Pareto-optimal *when the space of policies is restricted to DS policies*, but which are in fact *dominated* by a non-stationary (or stationary randomized (Chatterjee, 2007)) policy. To illustrate this, we adapt an example from the literature.

**Example 2.3.5** (Adapted from White (1982))**.** *Consider a discrete-time MDP with*

$\mathcal{S} = \{1\}, \mathcal{A} = \{1, 2, 3\}$, $p(1, a, 1) = 1$ *for all* $a \in \mathcal{A}$, *and*

$$\boldsymbol{c}(1, a) = \begin{cases} (3, 0), & a = 1, \\ (2, 2), & a = 2, \\ (0, 3), & a = 3. \end{cases}$$

*Let* $\mu_a$ *denote the DS policy satisfying* $\mu_a(1) = a$, *so that* $\mu_a$ *for all* $a$ *give the complete set of DS policies. If we let* $\boldsymbol{g}_a := \boldsymbol{g}(\mu_a)$, *then we easily see that* $\boldsymbol{g}_a = \boldsymbol{c}(1, a)$ *for all* $a$. *If we take the solution space to only be DS policies, then all of these policies are Pareto-optimal. However, if we define a randomised stationary policy* $\mu'$ *such that* $\mu'(1, 1) = \mu'(1, 3) = 1/2$ *and* $\mu'(1, 2) = 0$, *then at each step we uniformly randomly choose between the action that gives us* $(3, 0)$ *and the action that gives us* $(0, 3)$, *meaning in the limit we get* $\boldsymbol{g}(\mu') = (1.5, 1.5)$, *which dominates* $\boldsymbol{g}_2 = (2, 2)$. *Along similar lines, consider the deterministic non-stationary policy that alternates between decision rules* $\mu_1$ *and* $\mu_3$, *i.e., a policy* $\{d_t\}_{t \in \mathbb{Z}_{\geq 0}}$ *such that* $d_t = \mu_1$ *if* $t$ *is even, and* $d_t = \mu_3$ *otherwise. The sequences of one-step costs would then be* $\{(3, 0), (0, 3), (3, 0), (0, 3), ...\}$, *so that the finite-horizon cost-to-go satisfies* $\boldsymbol{J}_{2n-1} = (3n, 3n - 3)$ *and* $\boldsymbol{J}_{2n} = (3n, 3n)$ *making it easy to see that* $\boldsymbol{g}\left(\{d_t\}_{t \in \mathbb{Z}_{\geq 0}}\right) = \lim_{t \to \infty} \boldsymbol{J}_t = (1.5, 1.5)$.

It is shown by Chatterjee (2007) that all Pareto-optimal points can be obtained by using a randomised stationary policy, so going forward we can ignore non-stationary policies. We previously noted that the feasible solutions to the LP formulation for the CTMDP always correspond to a stationary distribution of an MRP induced by a stationary policy, so we can formulate a multi-objective CTMDP problem as a multi-

objective linear program:

$$
\begin{aligned}
&\min_{\pi} \sum_{(s,a)\in\mathcal{SA}} c_1(s,a)\pi(s,a) \\
&\quad\vdots \\
&\min_{\pi} \sum_{(s,a)\in\mathcal{SA}} c_K(s,a)\pi(s,a) \\
&\text{s.t.} \sum_{(s,a)\in\mathcal{SA}} q(s,a,s')\pi(s,a) = 0, \text{ for all } s' \in \mathcal{S}, \\
&\quad \sum_{(s,a)\in\mathcal{S}} \pi(s,a) = 1, \\
&\quad \pi(s,a) \geq 0 \text{ for all } s,a.
\end{aligned}
$$

We know that solving this using the mixed-objectives method will only yield basic feasible solutions, meaning solutions that correspond to Pareto-optimal DS policies. Pareto-optimal randomised stationary policies could be recovered from linear interpolations of these solutions; however, we argue that this is somewhat undesirable, due to the random and therefore unexplainable and untrustworthy nature of such a policy. We suggest that the goal of finding all DS policies that are Pareto-optimal with respect to the solution space of randomised stationary policies (i.e., basic feasible solutions to the LP under a mixed-objectives approach) qualitatively provides a good balance between the completeness of the front (and therefore the flexibility to the decision maker) and the interpretability and quality of the solutions.

It is also worth understanding the effect that the $\varepsilon$-constraint method has on the LP formulation. In general, adding inequality constraints that are linear in the state-action frequencies yields a *constrained* MDP, a topic thoroughly covered by Altman (2021). It is known in general that a constrained *unichain* MDP under the long-run average cost criterion with $L$ long-run average constraints has at most $L$ "randomisations" (Ross, 1989), where the number of randomisations for a randomised stationary policy $\mu$ is

$L(\mu) = \left(\sum_{s,a \in \mathcal{SA}} \mathbb{I}\{\mu(s,a) > 0\}\right) - |\mathcal{S}|$, i.e., the total number of actions that *could* be chosen for a state *in excess of* the one action that you must choose for a DS policy. As such, the $\varepsilon$-constrained version of a $K$-objective MDP could result in a policy with $K-1$ randomisations.

It is also worth demonstrating that the bi-objective LP formulation is not "well-behaved" for weakly-communicating LPs, due to the ability of the LP to essentially "choose its own starting distribution". We demonstrate this with an example of our own.

**Example 2.3.6.** *Consider a bi-objective CTMDP under the long-run average cost minimisation setting with* $\mathcal{S} = \{1, 2\}$, $\mathcal{A} = \{stick, swap\}$,

$$q(s, stick, s') = 0$$

*for all* $s, s'$ *(the stick action causes the state to be absorbing),*

$$q(s, swap, s') = \begin{cases} 1, & s \neq s', \\ -1, & s = s', \end{cases}$$

*and* $\boldsymbol{c}(1, stick) = (1, 0)$, $\boldsymbol{c}(2, stick) = (0, 1)$, *and* $\boldsymbol{c}(\cdot, swap) = (M, M)$ *for some large* $M >> 1$. *The availability of the very undesirable "swap" option makes this CTMDP weakly communicating. Pareto-optimal solutions are characterised by state-action frequencies of the form* $\pi(1, stick) = \alpha, \pi(2, stick) = 1 - \alpha$ *for* $\alpha \in [0, 1]$. *This corresponds to an "always stick" policy under initial distributions of the form* $\boldsymbol{\pi}_0^T = (\alpha, 1-\alpha)$, *and such a policy is clearly not unichain for* $0 < \alpha < 1$, *as in reality one would start in one state and just stay there.*

As such, the mixed-objectives method will always provide us with a DS policy that performs as expected irrespective of the starting distribution, whereas the $\varepsilon$-constraint

method does not have these guarantees. As such, when dealing with bi-objective MDPs (or problems that "contain" bi-objective MDPs as seen in later chapters), we simply use the mixed-objectives method.

This concludes our review of bi-objective optimisation for MILPs and MDPs and overall concludes the preliminary material. The following four chapters provide our novel contributions, which merge modelling and solution methodologies from deterministic optimisation (i.e., MILPs) and sequential optimisation under uncertainty (MDPs) in the context of the bi-objective design and/or dynamic (i.e., sequential) maintenance optimisation of systems with parallel redundancy. Chapter 3 reviews two prior frameworks for the integration of strategic and operational decision-making, and introduces our own general framework for tackling this problem. Chapter 4 considers the integrated design and maintenance optimisation of a parallel system, Chapter 5 investigates a special class of "decomposable" MDPs and uses this to model and optimise the maintenance of a series-parallel system, and Chapter 6 provides some initial work on the integrated design and maintenance optimisation of a series-parallel system.

# Chapter 3

# The Markov Decision Process Design Problem

## 3.1 Introduction

As noted in Section 1.1.1, a problem of interest is the integration of strategic and operational decisions in a single model. This way, strategic decisions can be jointly optimised alongside the dynamic operational decisions. We noted that, to the best of our knowledge, only three examples of an "MDP Design" problem have previously appeared in the literature, and that these examples have some shortcomings. In this chapter, we review these previous frameworks, and offer our own framework which builds upon one of the previous frameworks to allow for greater flexibility.

## 3.2 Existing Frameworks

In this section we discuss the three frameworks for MDP Design which are already in the literature. Section 3.2.1 discusses the work of Dimitrov and Morton (2009), and Section 3.2.2 discusses the work of Brown et al. (2024).

### 3.2.1 Action Set Design of Transient MDPs

Dimitrov and Morton (2009) consider an MDP Design problem in which first-stage design variables act as on-off switches for the permissibility of state-action pairs in a second-stage *transient* MDP under the total-reward criterion, where these first-stage design decisions must satisfy knapsack constraints. By "transient", we mean a type of MDP that always terminates in finite expected time for every policy, meaning it stops transitioning. Equivalently, one can imagine an MDP with an unavoidable absorbing state with no costs or rewards. Their model (up to some notational paraphrasing) is as follows:

$$\max_{\boldsymbol{z},\boldsymbol{x}} \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}(s)} r(s,a)x(s,a) \tag{3.2.1}$$

$$\text{s.t. } C\boldsymbol{z} \leq \boldsymbol{d}, \tag{3.2.2}$$

$$\sum_{a\in\mathcal{A}(s)} x(s,a) - \sum_{s'\in\mathcal{S}} \sum_{a\in\mathcal{A}(s')} p(s',a,s)x(s',a) = w_s, \text{ for all } s \in \mathcal{S}, \tag{3.2.3}$$

$$\sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}(s)} c(s,a)x(s,a) \leq b, \tag{3.2.4}$$

$$x(s,a) \leq M(s,a)z(s,a), \text{ for all } s,a, \tag{3.2.5}$$

$$x(s,a) \geq 0, \text{ for all } s,a, \tag{3.2.6}$$

$$z(s,a) \in \{0,1\}, \text{ for all } s,a. \tag{3.2.7}$$

The decision variables $z$ are binary decision variables indicating whether or not state-action pair $(s,a)$ is feasible in the resulting MDP, and $x(s,a)$ is a state-action occupancy measure, i.e., the total expected length of time that we expect the process to be in state $s$ taking action $a$ given starting distribution $w$ over the state space. Objective (3.2.1) is simply the total expected reward, expressed as the sum over the state-action pairs of their one-step rewards multiplied by the expected length of time spent in that state-action pair. Constraint (3.2.2) provides knapsack constraints over

the design variables, assuming that the entries of $C$ and $\boldsymbol{d}$ are non-negative (mild abuse of notions - read this as though $\boldsymbol{z}$ has been vectorised). Constraints (3.2.3) provide the balance equations for state-action occupancy. Constraint (3.2.4) is a constraint on total cost (with respect to some resource limited to $b$ units) accumulated over the time horizon. Constraint (3.2.5) is a "big-$M$" constraint that, for large enough choice of $M$, allows $x(s,a)$ to be essentially unbounded if $z(s,a) = 1$, or forced to 0 if $z(s,a) = 0$. Constraint (3.2.6) forces positivity on $x$, and constraint (3.2.7) forces $z$ to be binary.

With respect to this model, Dimitrov and Morton (2009) primarily speak of the decision variables $z(s,a)$ allowing for choices of actions for each state; however, the model presented is flexible enough to completely disallow states all together. A state $s$ could be avoided entirely if $w_s = 0$ and there exists a policy such that $s$ is inaccessible from all states $s'$ with $w_{s'} \neq 0$. However, this sort of structural prevention of states is not explored by the authors. Whilst being the first of its kind, the model suffers from many drawbacks: design variables are limited to knapsack-style constraints, there is a strict one-to-one correspondence between design variables and state-action pairs, and the model relies on big-$M$ constraints, for which the choice of such values is non-trivial, and where poor choices can lead to poor solutions (Brown et al., 2024). Nevertheless, this model does provide a foundation to build upon and generalise into a more flexible framework.

### 3.2.2 Cost Design of Discounted MDPs

In more recent work, Brown et al. (2024) consider an MDP Design problem in which the first-stage design variables are mixed-integer and only affect the one-step cost function of the second-stage infinite-horizon discounted MDPs. We use the plural because the second stage is the expectation over $|\Xi|$ possible scenarios for some finite scenario set $\Xi$. Specifically, each scenario has one-step cost functions for each state-action pair which are linear-affine in the design variables $x$. Their model is formulated as a *bilevel*

program, which is a mathematical program of the form:

$$\min_{\boldsymbol{x},\boldsymbol{y}} \left\{ \boldsymbol{c}_{\boldsymbol{x}}^T \boldsymbol{x} + \boldsymbol{c}_{\boldsymbol{y}}^T \boldsymbol{y} : A_{\boldsymbol{x}} \boldsymbol{x} = \boldsymbol{b}_{\boldsymbol{x}}, \boldsymbol{y} \in \arg\min \left\{ \boldsymbol{d}(x)^T \boldsymbol{y} : A_{\boldsymbol{y}}(\boldsymbol{x}) \boldsymbol{y} = \boldsymbol{b}_{\boldsymbol{y}}(\boldsymbol{x}) \right\} \right\},$$

such that some of the decision variables are constrained to be an optimal solution of a nested optimisation problem, the parameters of which are influenced by other decision variables (Dempe, 2002). They define their bilevel program as follows:

$$\min_{\boldsymbol{x},\boldsymbol{v}^\xi} \boldsymbol{c}^T \boldsymbol{x} + \sum_{\xi \in \Xi} q^\xi (\boldsymbol{\alpha}^\xi)^T \boldsymbol{v}^\xi \tag{3.2.8}$$

$$\text{s.t. } A\boldsymbol{x} = \boldsymbol{b}, \tag{3.2.9}$$

$$\boldsymbol{x} \in \mathcal{X}, \tag{3.2.10}$$

$$\boldsymbol{v}^\xi \in \operatorname*{argmax}_{\boldsymbol{v} \in \mathbb{R}^{|\mathcal{S}^\xi|}} \left\{ \mathbf{1}^T \boldsymbol{v} : v_s \leq (\boldsymbol{f}_{s,a}^\xi)^T x + g_{s,a}^\xi + \lambda_\xi \sum_{s' \in \mathcal{S}^\xi} p^\xi(s,a,s') v_{s'} \right.$$
$$\left. \text{for all } s \in \mathcal{S}^\xi, a \in \mathcal{A}^\xi \right\}, \text{ for all } \xi \in \Xi. \tag{3.2.11}$$

Objective (3.2.8) states that the goal is to minimise the sum of the cost of the design decisions ($\boldsymbol{c}^T \boldsymbol{x}$) and the expected total discounted cost of the second-stage MDPs, where this expectation is taken over the $|\Xi|$ scenarios (with probability $q^\xi$ for each scenario $\xi$) and initial state distributions $\boldsymbol{\alpha}^\xi$, and $\boldsymbol{v}^\xi$ is a vector such that $v_s^\xi$ is the total discounted cost of state $s$ in scenario $\xi$. Constraints (3.2.9) and (3.2.10) are simply generic linear constraints, and $\mathcal{X}$ describes a mixed-integer domain. Constraint (3.2.11) defines the second level (or "follower") problem, stating that $\boldsymbol{v}^\xi$ is the vector that satisfies the Bellman equations for scenario $\xi$. Note that this becomes a maximisation problem, because the LP version of the Bellman equations always takes on the opposite optimisation sense. The term $(\boldsymbol{f}_{s,a}^\xi)^T x + g_{s,a}^\xi$ describes the one-step costs of $(s,a)$ under scenario $\xi$, making it affine with respect to design variables $\boldsymbol{x}$. For each scenario $\xi$, $\lambda_\xi$ is a discount factor, $p^\xi$ are the transition probabilities, and $\mathcal{S}^\xi$ and $\mathcal{A}^\xi$ are the state and

action spaces, respectively.

The choice to model the problem as bilevel allows the problem to be compatible with pre-existing off-the-shelf bi-level solvers, and allows for the easy incorporation of scenario-based uncertainty in the resulting MDP. However, the restriction of the design variables to only affecting the one-step cost function is quite limiting, especially with respect to the ways in which different designs could impact the structure of the state-action space of any given MDP. To this end, the authors provide a reformulation of their bi-level model as a standard MILP, and they note that this model is more amenable to additional constraints to reflect the relationship between the designs and the resulting state-action space. However, their MILP model suffers from the same problem as Dimitrov and Morton (2009), as it requires a big-$M$ constraint for each state-action pair. The authors note that because of this, solving the MILP model using a MILP solver performs worse than solving the bi-level model using a bi-level solver.

### 3.2.3 Cost and Transitional Design of Discounted MDPs

In their PhD thesis, Liu (2022) introduces their *long-term strategic stochastic decision-making* (LSSD) framework, which is a non-linear program in which the mixed-integer first-stage design variables $x$ affect the transition probabilities and one-step reward functions of their second-stage discounted MDP, but the state spaces, action spaces, and discount factor remain constant. As such, this framework can be thought of as a slight generalisation of a single-scenario version of the framework of Brown et al. (2024). They also consider further extensions to constrained MDPs and constrained MDPs with variable budgets. They suggest discretising the design space to be approximately represented by a finite set of second stage MDPs (unfortunately, how they choose this discretization in such a way that they yield optimal solutions in their case study is unclear), and then solving this simplified model using a Benders-decomposition-based algorithm. The model and framework is then applied to the design and maintenance

of critical infrastructure networks against an attacker with imperfect information (so game-theoretical elements need not be considered). We omit the formulations, as no one formulation captures their whole framework and methodology. Like the previous two models, their model also suffers from the use of big-$M$ constraints, so these must be chosen carefully.

The following section presents a novel model for the MDP Design problem that bears more similarity to the model of Dimitrov and Morton (2009), whilst incorporating the interdependencies of the design variables and state-action pairs as suggested, but not implemented, by Brown et al. (2024).

## 3.3 The Binary-Linear Long-Run-Average MDP Design Framework

We assume throughout this subsection that all MDPs are in continuous-time (although a discrete-time equivalent could easily be produced). In general, a continuous-time $K$-objective MDP Design problem is a tuple:

$$\langle \mathcal{X}, \mathfrak{C} : \mathcal{X} \to \mathcal{MDP}, \boldsymbol{f} : \mathcal{XD} \to \mathbb{R}^K \rangle,$$

where $\mathcal{X}$ is a compact set of feasible designs, $\mathfrak{C}$ is a function that maps every design $x$ to a CTMDP $\mathfrak{C}(x)$, and $\boldsymbol{f}$ is a $K$-dimensional vector-valued function that maps every design-policy pair $(x, d) \in \mathcal{XD} := \{(x, d) : x \in \mathcal{X}, d \in \mathcal{D}(x)\}$, where $\mathcal{D}(x)$ is the set of all history-dependent randomised policies on $\mathfrak{C}(x)$, to a real vector. The goal is to find all Pareto-optimal design-policy pairs with respect to $\boldsymbol{f}$. Our aim is to define a set of reasonable assumptions such that an MDP Design problem that satisfies those assumptions can be represented as a Mixed-Integer Linear Program (MILP). That way, exact solutions can be found for small instances using MILP solvers, and the vast array

of theory on solving MILPs can potentially be used directly or otherwise adapted to solve this type of problem. To begin, we require a technical definition of an MDP contained within a larger MDP.

**Definition 3.3.1** (Sub-MDP)**.** *Let* $\mathfrak{C} = \langle \mathcal{S}, \mathcal{A}, q, \boldsymbol{c} \rangle$ *and* $\mathfrak{C}' = \langle \mathcal{S}', \mathcal{A}', q', \boldsymbol{c}' \rangle$ *be MDPs. We say that* $\mathfrak{C}'$ *is a* sub-MDP *of* $\mathfrak{C}$ *if the following are true:*

- $\mathcal{S}' \subset \mathcal{S}$,
- $\mathcal{A}'(s) \subset \mathcal{A}(s)$, *for all* $s \in \mathcal{S}'$,
- $q'(s, a, s') = q(s, a, s')$ *for all* $s, s' \in \mathcal{S}'$ *and* $a \in \mathcal{A}'(s)$,
- $q(s, a, s') = 0$ *for all* $s \in \mathcal{S}', a \in \mathcal{A}'(s), s' \notin \mathcal{S}'$ *(i.e., the sub-MDP cannot transition to a state not in the sub-MDP),*
- $\boldsymbol{c}'(s, a) = \boldsymbol{c}(s, a)$, *for all* $s \in \mathcal{S}', a \in \mathcal{A}'(s)$.

*If so, we write* $\mathfrak{C}' \leq \mathfrak{C}$.

This notion of a sub-MDP can be used to identify self-contained MDP sub-structures within larger MDPs. Of course, any MDP is a sub-MDP of itself. Another trivial example is that, for any stochastic stationary policy $\mu$ on an MDP $\mathfrak{C}$, there is a sub-MDP with $\mathcal{S}'$ given by the set of all recurrent states of $\mathfrak{C}$ under $\mu$, $\mathcal{A}'(s) = \{a : \mu(s, a) > 0\}$, and $q$ and $\boldsymbol{c}$ are simply carried over as expected. Sub-MDPs form part of the structural basis of our restricted MDP Design problems. Sub-MDPs also appear trivially when one defines the "disjoint union" of MDPs.

**Definition 3.3.2** (Disjoint Union of MDPs)**.** *Let* $\{\mathfrak{C}_i = \langle \mathcal{S}_i, \mathcal{A}_i, q_i, \boldsymbol{c}_i \rangle\}_{i=1}^{I}$ *be a finite indexed set of MDPs. The* disjoint union *of these MDPs is an MDP with the following elements:*

- *State Space* $\mathcal{S}_{\sqcup} = \bigsqcup_i \mathcal{S}_i = \{(i, s_i) : i \in [I], s_i \in \mathcal{S}_i\}$,

- *Action Spaces* $\mathcal{A}_{\sqcup}(s) = \mathcal{A}_i(s)$, *where* $i$ *is the label of* $s$,
- $q_{\sqcup}(s, a, s') = \begin{cases} q_i(s, a, s'), \textit{if } s \textit{ and } s' \textit{ have the same label } i, \\ 0, \textit{otherwise}, \end{cases}$
- $\boldsymbol{c}_{\sqcup}(s, a) = \boldsymbol{c}_i(s, a)$, *where* $i$ *is the index of* $s$.

Clearly, disjoint unions over MDPs provide a trivial way to construct multichain MDPs that contain the original MDPs as sub-MDPs (up to isomorphism on ignoring the labels). With sub-MDPs and disjoint unions defined, we may now define our restricted sense of an MDP Design problem.

**Definition 3.3.3** (Binary-Linear Long-Run-Average MDP Design problem)**.** *We say that an MDP Design problem* $\langle \mathcal{X}, \mathfrak{C}, \boldsymbol{f} \rangle$ *is* Binary-Linear Long-Run-Average *if it satisfies the following criteria:*

- $\mathcal{X}$ *is a bounded binary polytope, that is it can be expressed as:*

$$\mathcal{X} = \left\{ \boldsymbol{x} \in \{0,1\}^N : A^{ineq}\boldsymbol{x} \leq \boldsymbol{b}^{ineq}, A^{eq}\boldsymbol{x} = \boldsymbol{b}^{eq} \right\}$$

  *for constraint matrices* $A^{ineq}, A^{eq}$ *and constraint right-hand-sides* $b^{ineq}$ *and* $b^{eq}$.

- *There exists a* maximal *CTMDP* $\mathfrak{C}^{\max}$ *with finite state and action spaces such that* $\mathfrak{C}(\boldsymbol{x}) \leq \mathfrak{C}^{\max}$ *for all* $\boldsymbol{x} \in \mathcal{X}$.

- *For every* $s \in \mathcal{S}^{\max}, a \in \mathcal{A}^{\max}$, *there is a (feasibility) propositional formula* $FPF\{s, a\}$ *over the* $N$ *binary variables in conjunctive normal form (CNF, i.e., a conjunction/AND operator over clauses of disjunctions/OR operators over boolean variables and their negations) such that, if* $FPF\{s, a\}(\boldsymbol{x})$ *evaluates to true, then* $(s, a)$ *is a state-action pair in* $\mathfrak{C}(x)$.

- *Policies are restricted to being stationary.*

- *Each objective function $f_k$ is linear in $\boldsymbol{x}$ and $\boldsymbol{\pi}$, where $\boldsymbol{\pi}$ are the state-action frequencies. In keeping with the cost-rates $c_k$ of the maximal CTMDP, we say there exists a design cost vector $\boldsymbol{f}_k^{\boldsymbol{x}}$ such that $f_k(\boldsymbol{x}, \mu) = (\boldsymbol{f}_k^{\boldsymbol{x}})^T x + \sum_{s,a} c_k(s,a)\pi^\mu(s,a)$. As such, the MDPs are optimised with respect to the* long-run average *criterion*

- *There is no fixed initial state distribution.*

The above conditions essentially allow for a Binary-Linear Long-Run-Average-Cost MDP Design problem to be expressed as a large MILP, where binary variables or groups of binary variables act as "on-off switches" for groups of state-action pairs in a maximal CTMDP. We shall justify each assumption in turn:

- The assumption of a binary polytope for $\mathcal{X}$ is of course as general as is permissible for the binary variables if we are to aim for a MILP formulation.

- The assumption of a finite maximal CTMDP is always trivially true so long as $\mathcal{X}$ is finite, and $\mathfrak{C}(\boldsymbol{x})$ is finite for all $\boldsymbol{x}$ and , because then in the worst case $\mathfrak{C}^{\max}$ can be constructed by taking a disjoint union over the $\mathfrak{C}(\boldsymbol{x})$ for all $\boldsymbol{x}$. Of course, we would not expect such a choice to have any sort of exploitable structure to aid the optimisation or further the understanding of the problem.

- The assumption of propositional formulae in CNF for each state-action pair allows for each unique disjunctive clause to be associated with a linear constraint (we will demonstrate this shortly), while also maintaining flexibility as all propositional formulae can be expressed in CNF. This is also again trivially true following the disjoint union approach, as all state-action pairs in the union-derived maximal CTMDP corresponding to $\mathfrak{C}(\boldsymbol{x})$ can be given a CNF corresponding only to $\boldsymbol{x}$. Again, we would expect such a formulation to offer little in terms of interesting structure.

- The restriction to stationary policies is required by the standard LP formulations of an MDP. However, even for multi-objective MDPs, we know that stochastic

stationary policies are sufficient for covering the objective Pareto front for any given design.

- The limitation on the functional form of $\boldsymbol{f}$ forces it to be linear in $\boldsymbol{x}$, which is a requirement for the MILP framework, and forces the MDP objective to be the long-run average criterion with cost-rates $c_k$. The use of long-run average cost avoids the issues of big-$M$ constraints found in the previous two models.

- The final assumption removes problems related to multichain MDPs, which are out of the scope of this work. Even if an MDP is multichain, we imagine we can simply start it in the "best" recurrent class for any policy.

Before we complete our transformation of this problem into a MILP, we must first introduce some notation for the disjunctive clauses that make up the state-action propositional formulae. We denote the number of clauses in $FPF\{s,a\}$ by $M(s,a)$, and for each clause $m$, we define $N^+_{s,a,m}, N^-_{s,a,m} \subset [N]$ to be the subsets of indices such that the clause evaluates to true if $x_n = 1$ for some $n \in N^+_{s,a,m}$ or $x_n = 0$ for some $n \in N^-_{s,a,m}$. Of course, many of these clauses will be shared between the formulae of the state-action pairs. We define $\mathcal{C} = \bigcup_{s,a,m} \left\{(N^+_{s,a,m}, N^-_{s,a,m})\right\}$ to be the set of all unique clauses. We may now provide the MDP Design MILP as follows:

$$\min_{\boldsymbol{x},\boldsymbol{\pi}} (\boldsymbol{f}_1^{\boldsymbol{x}})^T \boldsymbol{x} + \sum_{(s,a)\in\mathcal{SA}^{\max}} c_1(s,a)\pi(s,a) \tag{3.3.1}$$

$$\vdots$$

$$\min_{\boldsymbol{x},\boldsymbol{\pi}} (\boldsymbol{f}_K^{\boldsymbol{x}})^T \boldsymbol{x} + \sum_{(s,a)\in\mathcal{SA}^{\max}} c_K(s,a)\pi(s,a) \tag{3.3.2}$$

$$\text{s.t. } A^{ineq}\boldsymbol{x} \leq \boldsymbol{b}^{ineq}, \tag{3.3.3}$$

$$A^{eq}\boldsymbol{x} = \boldsymbol{b}^{eq}, \tag{3.3.4}$$

$$\sum_{(s,a)\in\mathcal{SA}^{\max}} q^{\max}(s,a,s')\pi(s,a) = 0, \text{ for all } s' \in \mathcal{S}^{\max}, \tag{3.3.5}$$

$$\sum_{(s,a)\in\mathcal{SA}^{\max}} \pi(s,a) = 1, \tag{3.3.6}$$

$$\sum_{\substack{(s,a)\in\mathcal{SA}^{\max}:\\ FPF\{s,a\} \text{ requires}\\ \text{clause } (N^+,N^-)}} \pi(s,a) \leq \sum_{n\in N^+} x_n + \sum_{n\in N^-} (1-x_n), \text{ for all } (N^+,N^-) \in \mathcal{C}, \tag{3.3.7}$$

$$\pi(s,a) \geq 0, \text{ for all } s,a,$$

$$x_n \in \{0,1\}, \text{ for all } n \in [N].$$

Objectives (3.3.1) through (3.3.2) simply provide the most general linear form of an objective for this type of problem. Constraints (3.3.3) and (3.3.4) provide the polytopal constraints on the design decision space. Constraints (3.3.5) and (3.3.6) provide the balance equations for the maximal MDP. Constraints (3.3.7) ensure that the state-actions frequencies of all state-action pairs that require clause $(N^+, N^-)$ are forced to be 0 if the clause is not satisfied, and unrestrained by that particular constraint otherwise. Taken all together, all clauses of a state-action pair must be satisfied for the corresponding state-action frequency to be unconstrained from above. The final constraints then simply force positivity for the frequencies and force $\boldsymbol{x}$ to be binary.

We should compare our model to the previous two frameworks. Our formulation, the

Binary-Linear Long-Run-Average MDP Design framework, can best be thought of as a generalisation of the model considered by Dimitrov and Morton (2009), extending from transient MDPs to infinite-horizon MDPs under the long-run average cost criterion, and allowing for far more complicated relationships between the decision variables and the structure of the subsequent MDP. Note that all transient MDPs can be represented as long-run average MDPs under the stricter sense of bias optimality (Puterman, 2014, Chapter 10); however, our MILP formulation only considers gain optimality. It does this whilst remaining a MILP, and therefore could be tackled using the vast array of methods from that body of literature. The comparison against the model of Brown et al. (2024) is not so straight-forward. Our model is best suited to modelling problems where binary design variables deterministically impact the structure of the resulting MDP, and the use of long-run average cost helps to avoid the issues of big-$M$ constraints as seen in the other two models. The model of Brown et al. (2024) is better suited to problems with general design variables which only impact the cost structure of an MDP whose parameters and structure are uncertain.

## 3.4 Equivalence of MDP Design Problems to Standard MDPs

So far, we have presented each MDP Design formulation as a MILP or bi-level MILP. It is also worth noting that each formulation is also equivalent to a (very large or infinite-state) MDP. This likely isn't particularly helpful, but is worth including for completeness. All techniques essentially rely on the idea that the MDP equivalent of an MDP Design problem starts in a transient state in which the design decision is taken, which we refer to as a design state. In turn, we shall consider the formulations of Dimitrov and Morton (2009), Brown et al. (2024), Liu (2022), and our own.

First, we consider the model of Dimitrov and Morton (2009). Let $\mathcal{Z}$ be the set of

feasible design vectors, such that the knapsack constraints are satisfied and the MILP formulation is otherwise feasible (without this, some choices of $z$ may result in an invalid MDP). Let $\mathcal{S}' = \{0\} \cup \{(\boldsymbol{z}, s) : \boldsymbol{z} \in \mathcal{Z}, s \in \mathcal{S}\}$ be the state space of the MDP, ensuring that the state "remembers" the design decision taken, and state 0 represents the design state. Then let $\mathcal{A}'(\boldsymbol{z}, s) = \{a \in \mathcal{A}(s) : z(s, a) = 1\}$, $\mathcal{A}'(0) = \mathcal{Z}$ define the action spaces, so that the design suitably constrains the actions, and every design is accessible from state 0. We then let:

$$p'((\boldsymbol{z}, s), a, (\boldsymbol{z}, s')) = p(s, a, s'),$$
$$p(0, a, (\boldsymbol{z}', s')) = \begin{cases} w_{s'}, & \text{if } a = \boldsymbol{z}', \\ 0, & \text{otherwise,} \end{cases}$$

be the transition probabilities, $r'((\boldsymbol{z}, s), a) = r(s, a)$ be the one-step rewards and $c'((\boldsymbol{z}, s), a) = c(s, a)$ be the constraint costs. While we shall not prove it rigorously, it is hopefully reasonably clear that this MDP, when forced to start in state 0, is equivalent to the MDP Design problem of Dimitrov and Morton (2009) as presented in Section 3.2.1.

Next is the model of Brown et al. (2024). Recall that this model allows for scenario-based uncertainty in the resulting second-stage MDP. Let $\mathcal{X}$ be the mixed-integer polytope that represents all possible decisions, and note that this will be uncountably infinite if there is at least one continuous variable. We can define the state space of the MDP equivalent of this model as:

$$\mathcal{S}' := \{0, \infty\} \cup \left\{(\boldsymbol{x}, \xi, s) : \boldsymbol{x} \in \mathcal{X}, \xi \in \Xi, s \in \mathcal{S}^\xi\right\},$$

so that we have a "design" state 0, an absorbing state $\infty$ (needed due to scenario-dependent discount rates), and all other states "remember" the design taken and the scenario they're in. For actions, we have $\mathcal{A}'(0) = \mathcal{X}$, $\mathcal{A}'(\infty) = \{0\}, \mathcal{A}'(\boldsymbol{x}, \xi, s) = \mathcal{A}^\xi(s)$,

so that all design decisions are available in the design state, only one action is available in the absorbing state, and all other states have their suitable actions. For transition probabilities, we first set $p'(0, \boldsymbol{x}, (x, \xi, s)) = q^\xi \alpha_s^\xi$ for the transitions from the design state, and $p'(\infty, 0, \infty) = 1$ for the absorbing state. For other states, we set $p'((\boldsymbol{x}, \xi, s), a, (\boldsymbol{x}, \xi, s')) = \lambda^\xi p^\xi(s, a, s')$ and $p'((\boldsymbol{x}, \xi, s), a, \infty) = 1 - \lambda^\xi$, which ensures transitions follow the expected probabilities as per the original $p^\xi$, but with a $1 - \lambda^\xi$ stopping probability, which emulates the scenario-dependent discount rates. Finally, for costs, we have $c'(0, x) = \boldsymbol{c}^T \boldsymbol{x}$, $c'(\infty, 0) = 0$, and $c'((\boldsymbol{x}, \xi, s), a) = (\boldsymbol{f}_{s,a}^\xi)^T \boldsymbol{x} + g_{s,a}^\xi$. Under the expected total cost criterion, this model perfectly reflects the MDP Design problem of Brown et al. (2024) as long as it starts in state 0 with probability 1. Restricting to one scenario and suitably adjusting the transition probabilities and costs would yield an MDP equivalent for the framework of Liu (2022).

Finally, we consider our own model. We again let $\mathcal{X}$ be our binary design polytope. We can define the set of design-state pairs as follows:

$$\mathcal{XS} := \{(\boldsymbol{x}, s) \in \mathcal{X} \times \mathcal{S}^{\max} : \text{there exists } a \in \mathcal{A}(s) \text{ such that } FPF\{s, a\}(\boldsymbol{x}) \text{ is true}\}$$

For the state space of our equivalent MDP, we then set $\mathcal{S}' := \{0\} \cup \mathcal{XS}$, where the first part of the union is again a design state, and the second part gives all feasible design-state pairs. For action spaces, define $\mathcal{A}'(0) = \mathcal{X}$ and:

$$\mathcal{A}'(\boldsymbol{x}, s) = \{a \in \mathcal{A}(s) : FPF\{s, a\}(\boldsymbol{x}) = 1\}.$$

Due to the weakly communicating assumption and long-run average cost criterion, the design state transition rates $q'(0, \boldsymbol{x}, \cdot)$ need only satisfy $q'(0, \boldsymbol{x}, (\boldsymbol{x}, s)) = 0$ for all $(x, s) \notin \mathcal{XS}$ and $q'(0, \boldsymbol{x}, (\boldsymbol{x}', s)) = 0$ for all $\boldsymbol{x} \neq \boldsymbol{x}'$; it otherwise does not matter which state it transitions into next. The other transition rates then follow $q'((\boldsymbol{x}, s), a, (\boldsymbol{x}, s')) = q(s, a, s')$, and $q'((\boldsymbol{x}, s), a, (\boldsymbol{x}', s')) = 0$ for $x \neq x'$. Due to these transition rates, the

MDP equivalent is not itself weakly communicating, but multichain. Lastly, for the (multiple) objective functions, for each objective $k$ in the original MDP Design problem we set $c'_k(0, \boldsymbol{x}) = 0$ and $c'_k((\boldsymbol{x}, s), a) = (f_1^{\boldsymbol{x}})^T \boldsymbol{x} + c_k(s, a)$. This way, the long-run average cost will continuously reflect the upfront installation cost of the design. As set up, this MDP is equivalent to a Binary-Linear Long-Run-Average MDP Design problem when the initial state is the design state.

It is again worth repeating that these equivalences are "nice to know" mathematically, but not particularly useful. This is because these equivalent MDPs are very difficult to solve using standard MDP methods. These formulations would be far too large to be solved exactly by methods such as value iteration, policy iteration, or linear programming, especially with the MDP equivalent of Brown et al. (2024) being infinite in the state space. Approximate methods such as reinforcement learning are also unlikely to be useful, due to the transient nature of the design state, meaning this state is only learned about once per "episode" of learning despite being very important, and due to the lack of development of RL algorithms for multichain MDPs, especially in the long-run average case. Compared to the MILP formulations, which can have structure exploited by methods such as linear relaxations and can more readily be solved, standard methods for MDPs cannot make use of the structure of MDP Design problems.

Despite the fact that conversion to a standard MDP is not helpful in solving MDP Design problems, we can nevertheless use these insights and the commonalities across the four frameworks to provide an alternative and quite general definition for an MDP Design problem: *an MDP Design problem is simply an MDP (of any number of objectives and any type of objective sense) with a deterministic initial state 0 (i.e.,* $\pi_0(0) = 1$*) which cannot be returned to (i.e.,* $p(\cdot, \cdot, 0) \equiv 0$ *in the discrete-time case or* $q(\cdot, \cdot, 0) \equiv 0$ *in continuous time).* When the structure of $\mathcal{A}(0)$ is suitably complicated and impactful on the future of the process, especially if the process is strictly multichain, it then becomes fruitful to consider the conversion of this MDP into a MILP, bi-level MILP,

or something more exotic and suitably structured. This broad definition captures everything we need: an MDP Design problem is a sequential decision problem whose first decision is greatly impactful on the future of the process, potentially uncertain in impact (as per the formulation of Brown et al. (2024)), and cannot be revisited.

## 3.5 Conclusion

We have considered the limited previous literature on the MDP Design problem, and noted the strengths and weaknesses of these previous frameworks. We have also introduced our own framework for MDP Design and contrasted this against these two previous studies, taking note that our framework allows for far greater flexibility and modelling potential in the way that design decision can impact the *structure* of the resulting MDP. Where the prior literature has focused on MILP models, we have also shown that these MDP Design frameworks are also all equivalent to standard MDPs, but noted that this is not particularly useful due to the scale of the resulting equivalent MDPs.

In this thesis, we use our framework to model the simultaneous optimisation of the redundancy allocation (design) and maintenance (dynamic control) of parallel (Chapter 4) and series-parallel (Chapter 6) systems. We do *not* propose any exact solution methodologies for our proposed framework beyond standard MILP solvers, although we do discuss this as an avenue for future research in Section 7.2. Instead, we introduce a heuristic framework in Chapter 4 and demonstrate its applicability on the problem in that chapter, and we focus on a Dantzig-Wolfe decomposition approach in Chapter 6, which makes use of the heuristic methodology from Chapter 4. Chapter 5 focuses on a type of MDP that will be extended to include design decisions in Chapter 6.

# Chapter 4

# Bi-Objective Markov Decision Process Design for the Integrated Design and Dynamic Maintenance of a Parallel System

## 4.1 Introduction

Reliability is a feature of utmost importance in critical systems such as those in manufacturing, telecommunication, power generation and distribution, aircraft control, space exploration and satellite systems (Kim, 2018). Within such systems, no matter how complex, one can often identify a component or set of components that are critical to the operation of the system. These components together can be represented as a *series system*, or a system that fails overall if only one of its components fails. Reliability can also be linked to the operational costs of a system, where the failure of some component may require the usage of a less desirable and more costly alternative. It is a natural goal to improve the reliability of the system, and this can be achieved in various different

ways. One could consider investing in research and development to either manufacture new versions of critical components with higher component-wise reliability, or to entirely redesign the system such that the final result is more reliable overall. However, for many applications, particularly those using discrete component choices (e.g., most electronics), such an approach would be time-consuming, very costly, and would have diminishing returns. An alternative is to consider the introduction of redundancy into the system, whereby system-critical components are duplicated so that if one fails, the system can switch to using a duplicate component. This gives rise to the *redundancy allocation problem* (RAP).

In this work, we merge the design aspect of the RAP with a dynamic maintenance problem formulated as a Markov decision process (MDP) to create an *Integrated Design and Dynamic Maintenance Problem* (IDDMP). We believe this is the first MDP Design problem that both reflects an application and has non-trivial stochastic dynamics. Additionally, we consider the first bi-objective MDP (BO-MDP) Design problem, where to the best of our knowledge all prior work has been single-objective. On top of this, while most bi-objective RAP literature considers a second objective based on one-off installation costs, or total accumulated cost in finite mission time, our second objective is to minimize the long-run average cost over an infinite horizon, where these costs arise from usage and repair costs. Due to the complexity and novelty of the resulting model, we focus on single-subsystem problems in this chapter, as a first step towards application to more general models.

### 4.1.1 Redundancy Allocation Problem

In the broadest sense, the RAP is an NP-hard problem (Chern, 1992) in which some of the decision variables represent the installation of redundant components in some system, with knapsack-style constraints on the selected components (cost, weight, volume etc), and where the objective (or one of the objectives) is to maximize reliability,

or equivalently to minimize the probability of system failure. This is similar to but distinct from reliability allocation, where the structure (number and arrangement of components) is fixed, but each component has a range of alternatives from which to choose (Majety et al., 1999). For any component in the original design which is a candidate for further redundancy, we say that it may be replaced by a *subsystem* of parallel components. The RAP has seen many variations throughout the literature. It has seen single-objective (Chern, 1992) and multi-objective (Zoulfaghari et al., 2014; Kayedpour et al., 2017; Tavana et al., 2018; Lins and Droguett, 2011) variations, where secondary objectives often represent the installation costs of the components. The types of systems considered range between series-parallel systems (Chern, 1992) which only require one component in each subsystem to be working to fully function, $k$-out-of-$n$ systems which require some $k$ copies of each component to be operational simultaneously (Keshavarz Ghorabaee et al., 2015), and complex systems with complicated networked relationships between components (Park, 2020). Redundancy strategies may follow a hot, warm, or cold standby strategy, or a mixture of the three. Under hot standby, all components are considered to be *active* and therefore are subject to the same time-to-failure distributions as if they were being used. Under warm standby, all components are considered to be active, but with a lower rate of failure if they are not being used. Finally, under cold standby only the component currently being used is active, and the rates of failure for all other components are zero. When the active component fails, the system successfully switches to an idle component with some probability, where this is called *perfect switching* if the probability is always one. System failure occurs when both the active component fails and the switching fails. A mixed strategy keeps some components always active and others on standby (Reihaneh et al., 2022). Components may be modeled as all non-repairable (Kulturel-Konak et al., 2003), all repairable (Kayedpour et al., 2017), or a mix of the two (Zoulfaghari et al., 2014). The RAP may consider a finite *mission-time* (Kulturel-Konak et al., 2003; Kayedpour et al., 2017) or long-run

average reliability (Chern, 1992). On top of the standard allocation decisions, other decisions to be made may include the allocation of repair teams to each subsystem (Kayedpour et al., 2017; Lins and Droguett, 2011), or the order in which component switching occurs (Reihaneh et al., 2022). Components in the RAP may be binary-state, meaning that they are either fully working or fully damaged with no in-between, or they can be multi-state, allowing for a finer-grain representation of the state of repair of any given component (Tavana et al., 2018). The majority of the more recent literature uses metaheuristics such as genetic algorithms to optimize their respective models; however work has been done using Dantzig-Wolfe decomposition to create column-generation-based heuristics (Zia and Coit, 2010) or exact solution algorithms via branch-and-price (Reihaneh et al., 2022).

The specific type of RAP that we are interested in is a bi-objective repairable model for a series-parallel system. Zoulfaghari et al. (2014) proposes a bi-objective model to balance between cost and reliability, using a mixture of repairable and non-repairable components. Repairable components are actively maintained but the cost of performing these repairs is not considered. They optimize their model using a genetic algorithm. Kayedpour et al. (2017) proposes a bi-objective model that balances the availability of the system by a certain mission time against the combined cost of the components and of the repair workers allocated to each subsystem, where the number of repair workers allocated to each subsystem is an additional decision variable. Repairs are carried out actively (i.e., never delayed when repair workers are available), but a repair worker can only repair one component at a time. Component failure and repair times are assumed to be exponential. A *continuous-time Markov chain* (CTMC) model is used to model the stochastic dynamics of this system, and to evaluate its expected availability by the mission time. The model is optimized using NSGA-II, a genetic algorithm specific to multi-objective optimization. Tavana et al. (2018) also considers a bi-objective model balancing availability by a certain mission time and installation cost. This is a multi-

state model where every component goes through multiple states of degradation before failure. The dynamics of each component are again modeled using a CTMC. There is no limitation on the number of ongoing repairs, nor are there any associated costs. This model is also optimized using NSGA-II. Finally, Lins and Droguett (2011) considers a bi-objective model balancing reliability and cost; however, the cost term is more complicated. In addition to installation cost (called acquisition cost in their work), they also consider operational cost, corrective maintenance cost, repair team cost, and penalty for system failure. Here, operational costs are incurred per unit time from every healthy component under a warm standby model. Corrective maintenance costs are a lump sum paid every time a component is repaired, repair teams are paid per unit time, and there is a penalty incurred per unit time of system failure. Components are always repaired as soon as possible. The model is optimized using a multi-objective genetic algorithm alongside discrete event simulation.

Previous literature has explored the integration of the RAP for series-parallel systems with future maintenance decisions as a nonlinear two-stage stochastic programming problem with recourse. Bei et al. (2017) introduces such a two-stage problem where first-stage redundancy allocation decisions (allowing for different types of component per subsystem) are taken with uncertainty with respect to *future usage stresses*, which affect the assumed Weibull-distributed lifetimes of the installed components. After the random future usage stress has been realized, second-stage decisions then determine the *preventative maintenance interval* for each subsystem, which is the maximum length of time until a perfect preventative maintenance action is simultaneously and instantaneously applied to all components in a subsystem. If the subsystem fails before this point, an expensive emergency perfect repair must be performed. These perfect maintenance actions allow the stochastic dynamics of the system to be modelled as a renewal process, allowing for the straightforward calculation of expected maintenance costs per unit time, which in turn are used in the objective function (alongside

the installation cost of the components when adjusted by a capital inflation factor). Due to the nonlinear mixed-integer nature of the problem, the authors explore black-box methods (namely a commercial solver called NOMAD) to solve instances of their model, as well as integer rounding based on a continuous relaxation. Zhu et al. (2018) extend this by replacing the one-off usage stresses with stress *functions* that increase over time. The second-stage decisions determine (for each subsystem) the number of times minimal preventative repair will be performed, and how long each of the preventative maintenance intervals will be. First-stage decisions are simplified as there is only one component type per subsystem. To optimize their model, the authors use a decomposition approach where NOMAD is used to explore first-stage decisions, and some theoretical work allows the optimal second-stage decision for any given first stage to be solved by MATLAB's "fsolve" function. Bei et al. (2019) return to the simpler one-off stress formulation, and consider a more risk-averse problem formulation that introduces the conditional value-at-risk (CVaR) risk measure. Their solution methodology is similar, with NOMAD used to explore first-stage decisions and a combination of theoretical work and MATLAB's fsolve to optimize second-stage decisions. To the best of our knowledge, these papers are the only previous studies that integrate redundancy allocation with maintenance, and there is no literature on integrating redundancy allocation with dynamic maintenance. This is because, in general, there is very little literature on the integration of MDPs and first-stage design decisions into unified optimization models, let alone any literature that develops suitable, scalable solution methodologies.

Two recent, thorough literature reviews on RAP by Devi et al. (2023) and Guan et al. (2025) demonstrate that the problem is well established in the literature and very well studied. However, Devi et al. (2023) notes that the majority of the literature focuses on simple numeric examples not motivated by any real system, and there is a need to look towards real-world applications. Nevertheless, real-world applications have been considered by a number of authors, especially in the energy sector. Kuo (2001)

explores applications such as redundant electrical components in the control system of an aircraft, and redundant piping in a gas pipeline. Anand and Chidambaram (1994) considers redundancy allocation for a pressurized water reactor cooling loop. Süle et al. (2019) considers redundancy allocation with multiple objectives for safety-critical energy systems, directly considering the Sinopec Luoyang Petrochemical Plant in a case study. They consider balancing the number of redundant components against the overall risk of system failure. Ling et al. (2021) explores redundancy allocation for energy systems, with attention drawn to a case study considering biomass power plants. We note that these types of energy systems are repairable, and these repairs can be carried out whilst the functional components continue to operate. As such, the dynamic maintenance strategies that we explore in this work are applicable to these settings.

### 4.1.2 Markov Decision Processes

While some of the RAP literature includes Markov processes to model stochastic dynamics (Kayedpour et al., 2017; Tavana et al., 2018), to our knowledge none of the RAP literature makes use of MDPs. Common applications of MDPs include inventory management (Bertsekas, 2012), queueing control (Adusumilli and Hasenbein, 2010), and medical applications such as patient admission to hospitals (Nunes et al., 2009) and overflow decisions (Dai and Shi, 2019). MDPs have also been used to model maintenance problems, as small decisions (whether or not to repair, the extent of the repair, etc) can be made sequentially as the system degrades and is repaired over time. Some of this literature includes the condition-based preventative maintenance of systems, such as the work of Chen and Trivedi (2005) or Amari et al. (2006). More recent work considers *partially-observable MDPs* (POMDPs) for joint inspection and maintenance optimization (Guo and Liang, 2022) or maintenance planning alone (Deep et al., 2023). More relevant to the RAP is the application of MDPs to the maintenance of $k$-out-of-$n$

redundancy systems by Flynn and Chung (2004). Their work considers repairs with known deterministic lump-sum costs, a penalty for failure, and instantaneous repairs. The problem is modeled as a discrete-time MDP. Under these simplifying assumptions, they show that the optimal policy is a *Critical Component Policy* under reasonably light assumptions, meaning that every component is either always repaired straight away or never repaired at all, resulting in a very simple policy. However, we find that this does not hold for more the model proposed in this work. Similarly, Ozekici (1988) considers periodic age-based replacement of components in multi-component systems, and models this as an MDP.

### 4.1.3 MDP Design Problem

Rarely seen is the fusion of design decisions, such as those found in the RAP, with the dynamic operation of the system resulting from those design decisions, as modeled by an MDP. The main framework for this is the little-studied *MDP Design* problem. This is a two-stage problem in which the first stage decides upon the system design and the second stage optimizes the resulting system by modeling it as an MDP. To our knowledge, the earliest example of this framework is due to Dimitrov and Morton (2009), where for every state-action pair there is a vector cost associated with that state-action pair being available in the second-stage MDP. These vector costs must satisfy knapsack-style constraints. They also extend this model to include uncertainty in whether or not a design decision (i.e., allowing for a certain state-action pair) will truly be realized with that state-action pair being available. Dimitrov et al. (2013) apply this model to the problem of malaria intervention. An MDP Design framework was independently reformulated by Brown et al. (2024). In addition to formulating a *bilevel* problem where design decisions are taken in the first stage before operating the resulting MDP, this work also considers the possibility of uncertainty in the dynamics and costs of the associated MDP. For example, if a design decision reflects the installation of some

component in a system, there may be uncertainty in the reliability or costs associated with that component in the second stage. As such, they present a scenario-based approach to deal with this uncertainty. Suggested applications for this model include reliability, inventory management, and queueing design and control. They apply an off-the-shelf bi-level solver to a range of small randomly generated problems that do not reflect any specific application.

To the best of our knowledge, the three papers cited above form the entirety of the MDP Design literature as it stands, leaving much room for future development. We believe this represents an exciting opportunity, as MDP Design problems can be used to find improvements to the designs of real-world systems that will subsequently require sequential decisions to be made under uncertainty. In more general terms, this approach can allow for the integration of strategic and operational decisions in many different contexts. Relevant examples could include (i) an infrastructure system that must first be designed and deployed, then dynamically maintained as it experiences wear-and-tear over time; (ii) a service system (e.g., a hospital or transport hub) consisting of facilities that must first be designed and built before having their queues or waiting lists managed on a day-to-day basis; (iii) a dynamic variant of a location routing problem (Drexl and Schneider, 2015) in which depots must be strategically located before vehicle routes are managed on a daily basis (Pillac et al., 2013). Further development of the general area of MDP Design problems will help to address these sorts of important real-world problems.

### 4.1.4 Problem Overview

We present a high-level overview of our problem, aided by a simple example (the detailed problem formulation follows in Section 4.2). Consider a set of candidate components to be installed in parallel in a single subsystem. Each candidate component incurs usage cost at a constant rate when it is the component with the cheapest usage cost out of all

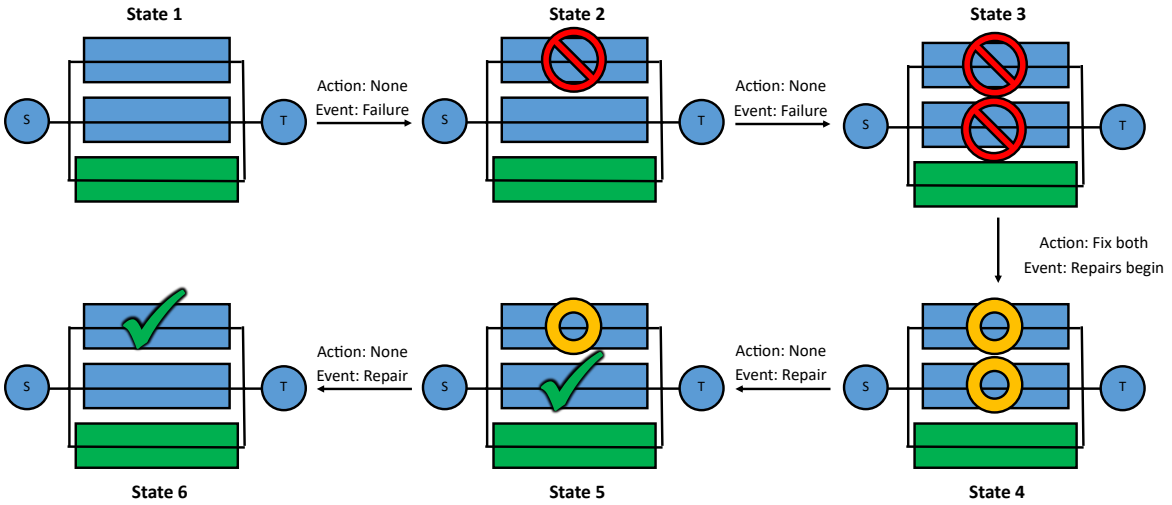



Figure 4.1.1: Sample trajectory through the dynamic maintenance problem.

healthy components. In other words, when the component we are currently using fails, we switch to a healthy component with the cheapest usage cost. Also, when a cheaper component comes back online, we switch back to using that component. We assume a hot standby strategy with perfect and instantaneous switching, with no or negligible idling costs. That is, usage costs are only incurred from the one component being used, and no costs are incurred from the idling-yet-active components. Each component also incurs repair costs at a constant rate while being repaired. These two costs contribute to the long-run operational costs, forming one of our objectives. The explicit modelling of usage costs in this way is a novel contribution to the RAP literature which allows us to, for example, model a type of "backup" component that is more reliable yet less power-efficient or less efficient in the processing of its inputs, making it more costly to run. Each component also has a failure rate and a repair rate corresponding to their respective failure and repair completion events. Finally, components also have attributes such as installation costs and weights, which are subject to constraints such as installation budgets and maximum total weights, respectively.

Our first-stage decisions are the quantities of each type of component to be installed. Our second stage decisions are dynamic maintenance decisions on which currently dam-

aged components to start repairing, given the current overall state of the subsystem. Figure 4.1.1 illustrates a sample trajectory through this model, considering a subsystem with three components, two of which are the same type, as indicated by the colors (available online). We start in state 1 with all components healthy and no maintenance decisions to be made, so we wait until the first failure event, which moves us to state 2. In this state, we can choose between two actions: fix this component, or do not fix this component. This is the type of situation where we may argue that it is unnecessary to repair this component straight away, since we have two further levels of redundancy. We assume that no repair action is taken, so we now wait until the next failure, taking us to state 3. Now, only one component remains healthy, so the next failure event would cause system failure. There are four possible actions, as each damaged component may or may not be put into repair. In this sample trajectory we decide to repair both, leading us to state 4. Now there are no further actions to take, and we simply wait for the next event, which is the completion of repairs on the second component in state 5. From state 5, there are still no actions to be taken, so we simply wait for the next event, which is the completion of repairs on the first component, leading us to state 6. State 6 is the same as state 1, with all components healthy. States 2 and 3 are most illustrative of the type of decisions we wish to make in the dynamic maintenance problem, where we want to decide which repairs to commence for any combination of components being healthy, damaged, or currently being repaired.

As a motivating example, consider the case of small-scale power generation. Suppose the owner of some system wishes to generate their own energy to power the system. Available to them are a range of generators all of which generate enough energy for the system, but which differ in terms of costs due to fuel consumption (either due to fuel costs or efficiency), maintenance costs, time-to-failure, repair time, installation cost, and so on. We assume that both long-run operational costs and reliability are important to the system owner, and that they have limits in terms of their upfront installation

budget and physical constraints such as weight and size, but that these constraints are relaxed enough to purchase and install multiple redundant generators for the sake of backup. We assume that these backup generators are kept in active (or hot) standby, i.e., they are kept idling (using a negligible amount of fuel) so that when one generator fails, an available generator with the cheapest usage costs can quickly and reliably take over. As such, the power generation fails only if all generators are simultaneously non-operational. Due to the presence of redundant generators, the owner is also interested in avoiding unnecessary maintenance costs, especially as they hope to operate this system well into the future, and maintenance will be needed repeatedly over a long period of time. On the other hand, the owner does not want to sacrifice too much in terms of reliability in the pursuit of saving on maintenance costs. As such, the problem is to simultaneously design a power system with redundancy and determine a maintenance strategy for it, in such a way as to strike a balance between two different objectives.

### 4.1.5 Our Contributions

In this chapter we make the following contributions:

- In Section 4.2, we introduce a novel formulation of an RAP for parallel systems as a bi-objective MDP Design problem, allowing for dynamic decisions to be made regarding maintenance. This is a new contribution to the RAP literature, and is also the first bi-objective MDP Design problem in the literature more generally. It is the first MDP Design formulation that allows for the feasibility of state-action pairs in the second-stage MDP to be affected by multiple first-stage design decisions, and for each first-stage design decision to impact the feasibility of multiple state-action pairs. This extends the formulation of Dimitrov and Morton (2009), which only allowed for a one-to-one correspondence between state-action pairs and design variables.

- In Section 4.2, we also propose the novel inclusion of usage costs incurred only when a component is being used, bearing some similarity to the way that costs are incurred in unreliable facility location problems (Lim et al., 2010, 2012) and unreliable $p$-median problems (O'Hanley et al., 2013). Alongside this, we consider component-wise (not repair-team-wise) repair costs.

- In Section 4.3, we introduce a heuristic for approximately solving bi-objective MDP Design problems, which is important due to the intractability of MDP Design problems of the form presented in Section 4.2.

- In Section 4.4, we verify the performance of our heuristic methodology against a standard exact MILP solver, and show that our heuristic performs well in terms of the quality and completeness of the Pareto fronts produced, whilst being much more computationally efficient. We also investigate the effects of varying parameters unique to our model, namely usage costs, repair costs, and the rates at which failure and repair events occur. This includes showing the benefits of using dynamic maintenance policies, which to our knowledge have never been considered in the RAP literature. Specifically, dynamic maintenance policies produce a better populated Pareto front, allowing better flexibility to decision-makers, and can also dominate solutions that are Pareto-optimal under the assumption of a non-dynamic policy.

Table 4.1.1 highlights our contributions against the relevant prior literature. We note from this table that our work is the only one to both be bi-objective and to include maintenance decisions other than repair team allocation, and our work is the only one to integrate dynamic maintenance decisions into the same optimisation model as the redundancy decisions. The structure of the chapter is as follows: in Section 4.2 we formulate the problem by formalizing the model shown in Figure 4.1.1 as a bi-objective CTMDP model, and then integrating it into a design problem. Section 4.3 builds

towards and explains our heuristic methodology for finding an approximate Pareto front of solutions to this problem. Section 4.4 then undertakes a computational study comparing our heuristic to the exact method, as well as exploring the effect of varying the parameters in our model that are less often seen in the RAP literature, such as usage costs, repair costs and event rates. Section 4.5 concludes, and proofs are deferred to the supplementary material.

| | Objective Type | | Maintenance Decisions | | | Operational Costs | | |
|---|---|---|---|---|---|---|---|---|
| Paper | Single | Bi | Repair Team Allocation | Static Schedule | Dynamic | Component-wise Maintenance | Component Usage | Solution Methodology |
| Lins and Droguett (2011) | | ✓ | ✓ | | | ✓ | ✓ | Genetic Algorithm with DES |
| Zoulfaghari et al. (2014) | | ✓ | | | | | | Genetic Algorithm |
| Kayedpour et al. (2017) | | ✓ | ✓ | | | | | NSGA-II |
| Bei et al. (2017) | ✓ | | | ✓ | | ✓ | | Black-Box; Integer Rounding |
| Tavana et al. (2018) | | ✓ | | | | | | NSGA-II |
| Zhu et al. (2018) | ✓ | | | ✓ | | ✓ | | Decomposition with black-box |
| Bei et al. (2019) | ✓ | | | ✓ | | ✓ | | Decomposition with black-box |
| This work | | ✓ | | | ✓ | ✓ | ✓ | Novel heuristic |

Table 4.1.1: Comparison of prior works and this work

## 4.2 Problem Formulation

In this section we formalize our problem by combining bi-objective optimization, continuous-time Markov decision processes and the MDP Design framework to provide a novel bi-objective CTMDP Design problem. Section 4.2.1 motivates our choice of using a bi-objective CTMDP to model this problem, describes the CTMDP in generality, then uses this framework to model our dynamic maintenance problem. Section 4.2.2 then extends this model to include design variables subject to knapsack-style constraints, and links the permitted states of the CTMDP to the design decisions made.

### 4.2.1 Dynamic Maintenance Problem

We first focus on how to model our *Bi-Objective Dynamic Maintenance Problem* (BO-DMP) as a CTMDP, assuming a fixed design initially. To begin, we discuss why bi-objectivity and dynamic maintenance are important to consider. In a single-objective formulation where the goal is just to maximize system reliability, the notion of dynamic maintenance—and the inclusion of operational costs—does not make sense, as the goal is only to improve system reliability, regardless of how frequent or expensive maintenance actions are. However, the solution to such a problem may be *overly* reliable if the constraints on the problem are quite relaxed, leading to an optimal system design that is very reliable, but costly to maintain and operate. A bi-objective formulation allows us to find a set of solutions that balance between operational costs and system reliability. More specifically, a Pareto front of solutions allows a decision maker to investigate how reasonable sacrifices in reliability can lead to a system with reduced long-run operational costs. This consideration of operational costs is also where the idea of using dynamic maintenance arises, as it leads to the question of whether repairing components immediately is always worthwhile, or whether it may sometimes be better to wait until the system is in a worse state overall. In the latter case, delays in repairs

and a less aggressive maintenance strategy may enable accumulated cost savings over the lifespan of the system. We now follow with the mathematical modelling of our problem.

We start by defining the CTMDP in general and introduce the relevant notation, and then define the states, actions, transitional behavior and costs of the BO-DMP. This is followed by a linear programming (LP) model which can easily be solved to optimality for realistic, moderately-sized instances.

Formally, an CTMDP is a 4-tuple:

$$\mathfrak{C} = \left\langle \mathcal{S}, (\mathcal{A}(s))_{s\in\mathcal{S}}, (q(s,a,s'))_{s,a,s'\in\mathcal{S},\mathcal{A}(s),\mathcal{S}}, (c(s,a))_{s,a\in\mathcal{S},\mathcal{A}(s)} \right\rangle,$$

where $\mathcal{S}$ is the state space, $\mathcal{A}(s)$ is the feasible action space for state $s \in \mathcal{S}$, $q(s,a,s')$ gives the transition rate from state $s$ to state $s'$ when taking action $a$, and $c(s,a)$ is the vector cost rate when taking action $a$ in state $s$. This definition is similar to that of Puterman (2014).

We restrict attention to deterministic stationary (DS) policies $\mu : \mathcal{S} \to \mathcal{A}$, giving maps from states to actions. For a fixed policy $\mu$, we can define the $T$-horizon cost-to-go or value function, where $T$ is some positive constant, $J_T^\mu : \mathcal{S} \to \mathbb{R}$:

$$J_T^\mu(s) = \mathbb{E}_\mu \left\{ \int_0^T c\,(\,S(t),\,\mu(S(t))\,)\,\mathrm{d}t \;\middle|\; S(0) = s \right\},$$

where $S(t)$ is a random variable representing the state of the process at time $t$. This represents the expected cost accumulated over $T$ units of time when starting the process in some state $s$ and controlling the process using policy $\mu$. Using this, we can then define the long-run average cost $g^\mu$ as follows:

$$g^\mu(s) = \lim_{T\to\infty} \frac{1}{T}\, J_T^\mu(s).$$

In the case of so-called *unichain* MDPs, the long-run average cost $g^{\mu}(s)$ under any policy $\mu$ is independent of the starting state $s$. Additionally, under the more relaxed assumption that the MDP is *weakly communicating*, the long-run cost $g^*$ under the optimal policy is independent of starting state (Puterman, 2014). We restrict our attention to such problems and later justify this with Theorem 4.2.1. In the single objective case, we wish to find a policy $\mu^*$ that minimizes $g^{\mu^*}$. In the multi-objective case, we wish to find non-dominated policies $\mu$ where there does not exist a policy $\mu'$ such that $g^{\mu'} \leq g^{\mu}$ component-wise and $g_k^{\mu'} < g_k^{\mu}$ for some entry (objective) $k$.

For the BO-DMP, we define the state $\boldsymbol{s} \in \mathbb{Z}^{N\times 2}$ of the CTMDP as an $(N \times 2)$-dimensional matrix representing the number of components of each type in either a repairing or damaged condition, where $N$ is the number of component types. Each row $\boldsymbol{s}_i$ of $\boldsymbol{s}$ is of the form

$$\boldsymbol{s}_i = [s_{i1}, s_{i2}],$$

where $s_{i1}$ is the number of components of type $i$ currently being repaired, and $s_{i2}$ is the number of components of type $i$ currently damaged and not being repaired. We assume that the total number of copies of component $i$ is known and given as $M_i$, so the number of healthy copies of component $i$ in any given state is $M_i - s_{i1} - s_{i2}$, which we denote by $s_{i0}$ for notational convenience. It is simple to see that $\boldsymbol{s}_i$ can take on $M_i(M_i + 1)/2$ possible values, as we require $s_{i1} + s_{i2} \leq M_i$. As such, the size of the state space is $|\mathcal{S}| = \prod_{i=1}^{N} M_i(M_i + 1)/2$, so our state space grows quadratically in the number of copies of any given component, and exponentially in the number of types of a component with at least 1 component installed.

The actions $\boldsymbol{a} \in \mathbb{Z}^N$ are vectors whose components $a_i$ are decision variables determining how many components of type $i$ to put into repair. For each state, we define the feasible action set $\mathcal{A}(\boldsymbol{s}) = \{\boldsymbol{a} \in \mathbb{Z}^N : 0 \leq a_i \leq s_{i2}\}$, as we can only repair components up to the number of damaged components of a specific type. For ease of notation, we write $\boldsymbol{s} \oplus \boldsymbol{a} \in \mathbb{Z}^{N\times 2}$ to denote the post-decision state (that is, the state immediately

after the action is taken, but before any new events are observed, following Powell (2011)), which satisfies $[\boldsymbol{s} \oplus \boldsymbol{a}]_i = [s_{i1} + a_i, s_{i2} - a_i]$. We assume that these actions are *impulsive*, meaning the effect of taking an action is instantaneous. For clarity, this does not mean that the repair itself is instantaneous, but that the process of starting a repair is instantaneous. We achieve this by enforcing the following relations when $\boldsymbol{a} \neq \boldsymbol{0}$:

$$
\begin{aligned}
q(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s'}) &= q(\boldsymbol{s} \oplus \boldsymbol{a}, \boldsymbol{0}, \boldsymbol{s'}), \text{ for } \boldsymbol{s'} \neq \boldsymbol{s}, \boldsymbol{s'} \neq \boldsymbol{s} \oplus \boldsymbol{a}, \\
q(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s} \oplus \boldsymbol{a}) &= 0, \\
q(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}) &= -\sum_{\boldsymbol{s'} \neq \boldsymbol{s}} q(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s'}), \\
c(\boldsymbol{s}, \boldsymbol{a}) &= c(\boldsymbol{s} \oplus \boldsymbol{a}, \boldsymbol{0}).
\end{aligned}
$$

This treats the state-action pair in such a way that it emulates the behavior of being in the post-action state under the zero action. Due to the separate treatment of non-zero actions, we can more succinctly write $q(\boldsymbol{s}, \boldsymbol{s'}) := q(\boldsymbol{s}, \boldsymbol{0}, \boldsymbol{s'})$ and $c(\boldsymbol{s}) := c(\boldsymbol{s}, \boldsymbol{0})$. This additional structure allows us to emulate the instant sojourn times of the impulsive MDPs of Dufour and Piunovskiy (2015), and we shall revisit this notion in greater detail in Chapter 5.

Outside of the immediate effects of actions, we also have two types of event which cause a state transition:

- **Failure** - The failure of a component, which occurs at a constant failure rate whenever the component is active. The interarrival times of such events are assumed to be independent and exponentially distributed with rate $\alpha_i > 0$ for component $i$.

- **Successful Repair** - The event that changes the condition of a repairing component so that it becomes a healthy component. We assume that ongoing repairs on each component are carried out simultaneously and independent of each other by

multiple repair workers, and therefore repair completion times are independent and exponentially distributed with rate $\tau_i > 0$ for component $i$.

We can then define the transition rates $q$ under the zero action as follows:

$$q(\boldsymbol{s}, \boldsymbol{s'}) = \begin{cases} s_{i0}\alpha_i, & \boldsymbol{s'} = \boldsymbol{s} + \boldsymbol{e}_{i2}, \\ s_{i1}\tau_i, & \boldsymbol{s'} = \boldsymbol{s} - \boldsymbol{e}_{i1}, \\ -\sum_{i=1}^{N} \left(s_{i1}(\tau_i + \alpha_i) + s_{i0}\alpha_i\right) & \boldsymbol{s'} = \boldsymbol{s}, \\ 0, & \text{otherwise}, \end{cases}$$

where $\boldsymbol{e}_{ij}$ is a matrix with dimensions the same as the state matrix with zeros everywhere except index $ij$, where it is 1. The first line gives the rate of a healthy component of type $i$ failing and becoming damaged, and the second line gives the rate of a repairing component of type $i$ completing its repair and becoming healthy. The third line then gives the overall negative sojourn rate of the state $\boldsymbol{s}$, and the fourth line shows that all other transitions have rate 0.

The cost rates of the BO-DMP should incorporate utilization costs, repair costs, and system availability. The cost rates are therefore made up of three components:

- **Component Utilization** - Each component $i$ has an associated cost $c_i$ per unit time to use the component, incurred only when it is the cheapest one available. The system is utilized optimally by always using the cheapest healthy component (recall that using a component does not increase its failure rate, so using the cheapest component does not increase the chance of needing to pay repair costs). Therefore we always incur usage costs $c^u(\boldsymbol{s}) = \min_{i:s_{i0}>0} \{c_i\}$.

- **System Failure** - If the system contains no healthy components, we incur a failure cost at rate 1 (i.e., this serves as an indicator variable for system failure).

- **Repair Costs** - If component $i$ is currently under repair then we incur a cost at rate $r_i$ per unit time. We denote this by $c^r(\boldsymbol{s}) = \sum_{i=1}^{N} r_i s_{i1}$.

Whilst utilization and repair costs can simply be added together due to being in comparable units, the availability of the system may not have a direct monetary value. As such, we treat system availability as a separate cost/objective, giving two separate objectives. Symbolically, we have an operational cost rate $c^o(\boldsymbol{s}) = c^u(\boldsymbol{s}) + c^r(\boldsymbol{s})$, and a failure cost rate $c^f(\boldsymbol{s}) = \prod_{i=1}^{N} \mathbb{I}\{s_{i0} = 0\}$.

We can model the BO-CTMDP using linear programming formulations, which when converted to a single objective problem can be solved using any LP solver. However, the LP formulation depends on certain properties of the CTMDP. The simplest LP model of an MDP is most applicable if the MDP is *unichain*, meaning that for any DS policy over the MDP, the induced Markov chain has at most one recurrent class. A more general quality is that of *communicating*, meaning that for every pair of states $s, s' \in \mathcal{S}$, there exists a policy $\mu$ that transitions from state $s$ to state $s'$ with non-zero probability in finitely many steps. A further generalization of this is *weakly communicating*, which allows for the existence of a (possibly empty) subset of states which are transient under every policy, and therefore are excluded from the choices of $s'$ in the original condition. For weakly communicating MDPs, we can use the unichain LP formulation to obtain an optimal solution, with the caveat that the solution will not specify actions for the transient states under this optimal policy. A thorough explanation of the differences between optimizing unichain and multichain MDPs is given by Puterman (2014). We provide a theoretical result with Theorem 4.2.1, demonstrating that the MDP we have formulated is weakly communicating, and thus the simpler LP formulation is suitable.

**Theorem 4.2.1.** *DMP is weakly communicating for all $N > 0$. However, there exist $N$ and $M_i$ for $i = 1, ..., N$ such that DMP is not unichain.*

*Proof.* See Section A.1.1. □

As a result of Theorem 4.2.1, we can use the most simple LP formulation of the CTMDP, following Guo and Hernández-Lerma (2009). Our first formulation is called

the *Bi-Objective Dynamic Maintenance Problem*, or BO-DMP:

$$\text{(BO-DMP)} \qquad \min_{\pi} g^o = \sum_{\boldsymbol{s}\in\mathcal{S}} \sum_{\boldsymbol{a}\in\mathcal{A}(\boldsymbol{s})} c^o(\boldsymbol{s},\boldsymbol{a})\pi(\boldsymbol{s},\boldsymbol{a}) \tag{4.2.1}$$

$$\min_{\pi} g^f = \sum_{\boldsymbol{s}\in\mathcal{S}} \sum_{\boldsymbol{a}\in\mathcal{A}(\boldsymbol{s})} c^f(\boldsymbol{s},\boldsymbol{a})\pi(\boldsymbol{s},\boldsymbol{a}) \tag{4.2.2}$$

$$\text{s.t.} \sum_{\boldsymbol{s}\in\mathcal{S}} \sum_{\boldsymbol{a}\in\mathcal{A}(\boldsymbol{s}')} q(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}')\pi(\boldsymbol{s},\boldsymbol{a}) = 0,\ \forall \boldsymbol{s}' \in \mathcal{S}, \tag{4.2.3}$$

$$\sum_{\boldsymbol{s}\in\mathcal{S}} \sum_{\boldsymbol{a}\in\mathcal{A}(\boldsymbol{s})} \pi(\boldsymbol{s},\boldsymbol{a}) = 1, \tag{4.2.4}$$

$$\pi(\boldsymbol{s},\boldsymbol{a}) \geq 0,\ \forall \boldsymbol{s} \in \mathcal{S}, \boldsymbol{a} \in \mathcal{A}(s). \tag{4.2.5}$$

Each decision variable $\pi(\boldsymbol{s},\boldsymbol{a})$ in the above formulation is a state-action frequency, which represents the proportion of time spent in state $\boldsymbol{s}$ and taking action $\boldsymbol{a}$. Objectives (4.2.1) and (4.2.2) are the operational costs and system failure objectives, respectively. Constraints (4.2.3) are the CTMDP equivalent of the well-known balance equations associated with standard CTMCs. Constraints (4.2.4) and (4.2.5) ensure that $\pi$ is a valid probability distribution. This rather general formulation of a bi-objective CTMDP requires additional assumptions that we are using a stationary policy, and that the CTMDP is weakly communicating. While we have proven the latter for our case, we must make the argument for the former. While it is known that DS optimal policies always exist for the single-objective case, and therefore attention can be limited to such policies, the same doesn't hold for multi-objective MDPs. Following White (1982), non-stationary policies may offer Pareto-optimal solutions that strike a unique balance between objectives that cannot be emulated by DS policies. However, their unique objective values can be emulated by a stochastic stationary policy (Roijers et al., 2013). These Pareto-optimal stochastic policies will correspond to points on the facets of the polytope defined by (4.2.3)-(4.2.5), which in turn means they can be interpreted as stochastic interpolations of the basic feasible solutions (i.e., DS policies). However,

in practice, these stochastic policies may be considered unintuitive, untrustworthy, or difficult to interpret. This is especially the case for areas such as reliability, where the suggestion of making random maintenance decisions could be treated with suspicion. Additionally, for problems of even a moderate size, the DS policies alone provide a good amount of flexibility with respect to balancing between the two objectives. As such, we will limit ourselves to finding DS policies that are non-dominated with respect to stochastic stationary policies, so that DS policies that are dominated by some stochastic stationary policy will be ignored. To ensure that we only find such solutions, we will scalarize our bi-objective program using the mixed objectives method, yielding a single-objective problem with no additional constraints. As such, a solution to a scalarized problem as returned by a DP method or LP solver will be a basic feasible solution, i.e., a DS policy. To do this, we introduce a penalty parameter $p$ to introduce a virtual monetary cost associated with system failure, and add this to our first objective. This yields the problem we call $p-$DMP:

$$(p-\text{DMP}) \qquad \min_{\pi} g = \sum_{\boldsymbol{s}\in\mathcal{S}} \sum_{\boldsymbol{a}\in\mathcal{A}(\boldsymbol{s})} \left(c^o(\boldsymbol{s},\boldsymbol{a}) + pc^f(\boldsymbol{s},\boldsymbol{a})\right)\pi(\boldsymbol{s},\boldsymbol{a})$$

$$\text{s.t. } (4.2.3) \text{ - } (4.2.5).$$

This problem provides a scalarized version of BO-DMP, and is equivalent to a weighted sum single objective. We can solve this problem for different values of $p$ to obtain a set of efficient solutions that correspond to DS policies. As $p-$DMP is a single-objective LP, it can be solved easily using any LP solver as long as the state-action space is of a manageable size.

### 4.2.2 Integrated Design and Dynamic Maintenance Problem

In the *Integrated Design and Dynamic Maintenance Problem* (IDDMP), we wish to simultaneously optimize both the design variables of the redundant system and the dynamic policy used to maintain it, with respect to knapsack-style constraints $A\boldsymbol{x} \leq \boldsymbol{b}$ (with all entries in $A$ and $b$ non-negative). To do this, we must first construct a state space that contains the maximum number of copies of each component with respect to the constraints:

$$\mathcal{S}^{\max} = \bigtimes_{i=1}^{N} \mathcal{S}_i,$$

where $\mathcal{S}_i = \{(s_1, s_2) \in \mathbb{N}_0^2 : s_1 + s_2 \leq M_i\}$ gives the maximal one-component state space for component $i$ and $M_i = \min_i\{b_j/a_{ji}\}$ gives the maximum number of copies of component $i$ with respect to the knapsack constraints. Using this, we may extend $p$-DMP to an *MDP Design problem* which includes design decision variables as follows:

$$(p\text{–IDDMP}) \quad \min_{\boldsymbol{\pi},\boldsymbol{x}} g = \sum_{\boldsymbol{s}\in\mathcal{S}^{\max}} \sum_{\boldsymbol{a}\in\mathcal{A}(\boldsymbol{s})} \left(c^o(\boldsymbol{s},\boldsymbol{a}) + pc^f(\boldsymbol{s},\boldsymbol{a})\right)\pi(\boldsymbol{s},\boldsymbol{a}) \tag{4.2.6}$$

$$\text{s.t. (4.2.3) - (4.2.5)} \tag{4.2.7}$$

$$A\boldsymbol{x} \leq \boldsymbol{b} \tag{4.2.8}$$

$$\sum_{\boldsymbol{s}\in\mathcal{S}^{\max},\boldsymbol{a}\in\mathcal{A}(s):s_{i2}-a_i\leq M_i-j} \pi(\boldsymbol{s},\boldsymbol{a}) \leq x_{ij} \text{ for } i = 1,2,...N,\ j = 1,...,M_i \tag{4.2.9}$$

$$x_{i(j+1)} - x_{ij} \leq 0 \text{ for } i = 1,...,N,\ j = 1,...,M_i - 1 \tag{4.2.10}$$

$$x_{ij} \in \{0,1\} \text{ for } i = 1,2,...N,\ j = 1,...,M_i. \tag{4.2.11}$$

We introduce binary variables $x_{ij}$ to represent our design decisions, where $x_{ij} = 1$ represents the decision to install the $j$th copy of component $i$. Objective function (4.2.6) and constraints (4.2.7) are carried over from $p$-DMP, with $\mathcal{S}$ replaced by $\mathcal{S}^{\max}$. Constraints (4.2.8) are the knapsack-style constraints over the design variables. Constraints

(4.2.9) ensure that the probability of having more than $j$ healthy or repairing copies of component $i$ is limited by $x_{ij}$. This means that the probabilities are unconstrained if $x_{ij} = 1$, or are all forced to be zero if $x_{ij} = 0$. This works on the principle that having some number of damaged components that cannot ever be repaired is equivalent to not having those components at all. Constraints (4.2.10) enforce the logic that if you do not install the $j$th copy of component $i$, then you cannot install the $(j+1)$th copy. Constraints (4.2.11) ensure that the $x_{ij}$ variables are binary. We note that BO-DMP can be extended in a similar way to obtain BO-IDDMP, but we omit the formulation for brevity.

While the formulation of $p$-IDDMP appears to be quite general and applicable to other problems, we must take note of its limitations. It requires that the large underlying MDP over the state space $\mathcal{S}^{\max}$ is weakly communicating, and one would have to adapt the multichain LP of Puterman (2014) to the design setting to get a more general framework, which may require additional care. Also, our model relies on the assumption that we are in the long-run average optimality setting. In the discounted setting, the sum of the variables is bounded above by $M_\gamma = 1/(1-\gamma)$, where $\gamma$ is the discount factor. This large constant would have to be integrated into constraint (4.2.9), leading to a far weaker constraint in the linear relaxation. Further discussion of a variant of this big-M constraint in the context of discounted MDP Design problems is given by Brown et al. (2024). We recognize that $\mathcal{S}^{\max}$ grows quadratically in the number of allowed copies of any given component, and grows exponentially in the number of component types. As such, the number of continuous state-action frequency variables $\pi$ and the number of equality constraints from the balance equations grow very rapidly with the problem size (more specifically, the number of component types and the leniency of the knapsack constraints). However, many state-action pairs will be infeasible under any design, as they represent a design that violates the knapsack constraints. As such, a more careful construction of $\mathcal{S}^{\max}$ and of $\pi$ will yield a much reduced set of variables.

| Notation | Description |
|---|---|
| CTMDP | |
| $\boldsymbol{s}$ | State vector |
| $\boldsymbol{a}$ | Action vector |
| $\mu$ | Policy |
| $g_\mu$ | Long-run average cost under policy $\mu$ |
| $q(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s'})$ | Transition rate from state $\boldsymbol{s}$ to $\boldsymbol{s'}$ under action $\boldsymbol{a}$ |
| $c(\boldsymbol{s})$ | Cost rate of state $\boldsymbol{s}$ |
| Problem Parameters | |
| $N$ | Number of types of component |
| $M_i$ | Maximum number of copies of component type $i$ |
| $\alpha_i$ | Rate of failure for component $i$ |
| $\tau_i$ | Rate of repair for component $i$ |
| $c_i$ | Usage cost rate of component $i$ |
| $r_i$ | Repair cost rate of component $i$ |
| $p$ | The penalty incurred for system failure |

Table 4.2.1: Table of notation

Further details can be found in Section A.1.2. The problem is also complicated by the presence of binary decision variables, leading to a very large-scale mixed integer linear program (MILP). As such, we cannot hope to solve this problem directly for problems even of moderate size, and must design some bespoke solution methodology. Section 4.3 follows with the introduction of our heuristic methodology for approximating a set of Pareto-optimal solutions for BO-IDDMP. The notation introduced in this section is summarized in Table 4.2.1.

## 4.3 Methodology

In this section we present a novel yet simple heuristic solution methodology for BO-MDP Design problems. Section 4.3.1 introduces the method, which we call the Approximate Pareto Population (APP) method, in full generality, and provides a simple bound on performance. Section 4.3.2 provides an interpretation of how this algorithm works in

general and in context of our problem, and explores how it might be applicable to other problems. Section 4.3.3 analyses the main sub-problem that arises when we apply APP to BO-IDDMP, and Section 4.3.4 gives an overview and illustration of the algorithm.

### 4.3.1 Approximate Pareto Population

APP is a novel heuristic framework for BO-MDP Design problems. It is a population- and decomposition-based method that breaks down a BO-MDP Design problem into smaller sub-problems. We provide the method in full generality here as it could be applicable to other problem settings as a heuristic or benchmark, however the remainder of the chapter will focus on its application to BO-IDDMP. The general form for a BO-MDP Design problem can be represented as a master problem MP:

$$\begin{aligned} \text{(MP)} \quad & \min_{\boldsymbol{x},\mu} f^1(\boldsymbol{x},\mu) \\ & \min_{\boldsymbol{x},\mu} f^2(\boldsymbol{x},\mu) \\ & \text{s.t. } \boldsymbol{x} \in \mathcal{X}, \mu \in \mathcal{M}(x). \end{aligned}$$

Here, $\boldsymbol{x}$ is the decision vector with feasible region $\mathcal{X}$, and we assume that for each decision there is an associated MDP (or CTMDP) $\mathfrak{C}(\boldsymbol{x})$ over which there is a full space of (potentially stochastic and non-stationary) policies $\mathcal{M}(x)$. The two objective functions $f^1$ and $f^2$ both take the design and policy as inputs.

The key idea of APP is that we first generate a finite set of candidate designs $\mathcal{X}^*$ which strike different balances between the two objectives. Suppose that for each design $\boldsymbol{x}$ we have an associated set of heuristic solutions $\mathcal{H}(x) \subset \mathcal{M}(x)$. By restricting MP to

these heuristic solutions, we obtain our first subproblem SP1.

$$\begin{aligned}(\text{SP1}) \quad & \min_{\boldsymbol{x},\mu} f^1(\boldsymbol{x},\mu) \\ & \min_{\boldsymbol{x},\mu} f^2(\boldsymbol{x},\mu) \\ & \text{s.t. } \boldsymbol{x} \in \mathcal{X}, \mu \in \mathcal{H}(\boldsymbol{x}).\end{aligned}$$

The quality of solutions found by SP1 is of course determined by the chosen heuristics $\mathcal{H}(x)$. Based on the quality of these heuristics, we can bound the performance of SP1.

**Proposition 4.3.1.** *Let* $f_w(\boldsymbol{x},\mu) = wf^1(\boldsymbol{x},\mu) + (1-w)f^2(\boldsymbol{x},\mu)$ *for* $w \in [0,1]$*, and suppose that for all* $\boldsymbol{x}$ *and* $w$*,* $\mathcal{H}(\boldsymbol{x})$ *is* $(1+\varepsilon)$*-optimal for* $\min_{\mu\in\mathcal{M}(\boldsymbol{x})}\{f_w(\boldsymbol{x},\mu)\}$*, i.e.,* $\min_{\mu\in\mathcal{H}(\boldsymbol{x})}\{f_w(\boldsymbol{x},\mu)\} \le (1+\varepsilon)\min_{\mu\in\mathcal{M}(\boldsymbol{x})}\{f_w(\boldsymbol{x},\mu)\}$*. Then the optimal solution of SP1 is* $(1+\varepsilon)$*-optimal for MP.*

*Proof.* Let $(\boldsymbol{x}^*,\mu^*)$ be an optimal solution for $\min_{\boldsymbol{x}\in\mathcal{X},\mu\in\mathcal{M}(\boldsymbol{x})}\{f_w(\boldsymbol{x},\mu)\}$ (i.e., the mixed-objective version of MP). Then there exists a $\mu \in \mathcal{H}(\boldsymbol{x}^*)$ such that $f_w(\boldsymbol{x}^*,\mu) \le (1+\varepsilon)f_w(\boldsymbol{x}^*,\mu^*)$, and this $(x^*,\mu)$ is feasible for SP1. Therefore the optimal solution of SP1 is $(1+\varepsilon)$-optimal for MP under a mixed-objective scalarization. □

The design solutions of SP1 form a non-dominated front that we denote by $\mathcal{X}^*$. This set could in general be infinite if $\mathcal{X}$ is infinite (e.g. if $\mathcal{X}$ contains continuous variables). However, we focus on the finite case here. The second stage of APP iterates over all candidate designs $\boldsymbol{x} \in \mathcal{X}^*$ and produces a Pareto front of solutions with the decision variables fixed. For any first-stage decision $\boldsymbol{x}$, we call this sub-problem SP2($\boldsymbol{x}$).

$$\begin{aligned}(\text{SP2}(\boldsymbol{x})) \quad & \min_{\mu\in\mathcal{M}(\boldsymbol{x})} f^1(\boldsymbol{x},\mu) \\ & \min_{\mu\in\mathcal{M}(\boldsymbol{x})} f^2(\boldsymbol{x},\mu)\end{aligned}$$

We denote the Pareto front of SP2($x$) by $\mathcal{M}^*(x)$. SP2 both closes the $(1+\varepsilon)$-optimality gaps of the solutions found in SP1, and also explores how well other design solutions perform for different weightings of the two objectives, (or, in other words, explores different parts of the Pareto front). However, this is not always guaranteed to yield improvements.

**Remark 4.3.1.** *Consider an absolute worst-case scenario where, for each $w$, $\mathcal{H}(\boldsymbol{x})$ attains the $(1+\varepsilon)$ bound for the true optimal design $\boldsymbol{x}^*_w$, and SP1 finds a solution $(\boldsymbol{x}_w, \mu_w)$ such that $\mu_w$ is truly optimal for $\boldsymbol{x}_w$ but attains the same value as found for the optimal design, i.e., $f_w(\boldsymbol{x}_w, \mu_w) = (1+\varepsilon) f_w(\boldsymbol{x}^*_w, \mu^*_w)$. As solvers generally only return one of the optimal solutions, in the worst case the sub-optimal design $\boldsymbol{x}_w$ is returned for each value of $w$ instead of $\boldsymbol{x}^*_w$, which cannot be improved by SP2 for the weight $w$.*

Of course, the scenario described in Remark 4.3.1 is a worst-case scenario that one would not consider to be at all common, but it does imply that we cannot make any claims about guaranteed improved bounds from SP2. However, it is clear that whenever SP2 can be easily solved, it makes sense to try it. Even in cases where SP2 is not easily solvable, it is still worth applying approximate methods such as ADP or RL to seek improvements on the solutions found in SP1.

Our overall population of candidate solutions can then be denoted as $P = \{(\boldsymbol{x}, \mu) : \boldsymbol{x} \in \mathcal{X}^*, \mu \in \mathcal{M}^*(x)\}$. Our final approximate Pareto front can then be determined as the set of non-dominated solutions in $P$ with respect to the objectives $f^1$ and $f^2$. Despite the somewhat weak performance guarantees of this methodology, especially with respect to SP2, we believe that this framework is an important first step to obtaining good solutions to an otherwise hard and novel class of integrated problems, and can comfortably serve as a heuristic in its own right, or a benchmark against which future methods can be compared. With our general methodology outlined, we now offer an interpretation of this algorithm.

### 4.3.2 Interpretation and Applicability

In the case of solving BO-MDP Design problems, APP first solves a bi-objective design problem that is restricted to a set of heuristic policies $\mathcal{H}(\boldsymbol{x})$ to control the sequential problem, and then solving the true dynamic problem using the designs found in the first stage. We acknowledge that the problems SP1 and SP2 may be challenging to solve in their own right, and could require approximate methods such as metaheuristics or reinforcement learning, respectively. By design, the algorithm does of course produce a non-dominated set of solutions (with respect to itself) that interpolate between the two single-objective heuristic solutions, and a real-world decision maker can ultimately decide whether or not the solutions provide a good trade-off between the objectives for real-world deployment, regardless of whether or not they are truly Pareto-optimal.

We now turn our attention back to the IDDMP. A reasonable choice for a set of heuristic policies is those policies that always repair some subset of the installed components as soon as they become damaged, and entirely neglect the other components, along the same lines as the critical component policies of Flynn and Chung (2004). However, in our problem, finding a design $x$ and a critical component policy $\mu$ that always repairs a subset of the components represented by $x$ will yield an identical outcome to choosing a design $x'$ that represents the components actively maintained by $\mu$, and choosing to actively maintain all components. As such, for any design $x$, we simplify the problem by allowing only one heuristic maintenance policy (so $|\mathcal{H}(\boldsymbol{x})| = 1$), which is the one that always maintains all active components. We can then interpret SP1 as finding different combinations of components that strike different balances between operational costs and system reliability, under the assumption that all components are actively maintained. Then, for each design $\boldsymbol{x}$ that comes out of this, SP2($\boldsymbol{x}$) will find Pareto-optimal dynamic maintenance policies for design $\boldsymbol{x}$. It may sometimes happen that when we have two designs $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ such that $\boldsymbol{x}^{(1)}$ performs better than $\boldsymbol{x}^{(2)}$ with respect to the first objective (cost) but worse than $\boldsymbol{x}^{(2)}$ with respect to the sec-

ond objective (reliability) under the fully active maintenance policy, the design $\boldsymbol{x}^{(2)}$ may then yield a 'lazier' maintenance policy that performs better than the fully active policy under $\boldsymbol{x}^{(1)}$ with respect to both objectives. That is to say a lazy maintenance policy on an intrinsically more reliable design may be both cheaper and more reliable in the long-run than the fully active maintenance on a cheaper, less reliable system. This shows the benefit of heuristically solving the problem in one large bi-objective sweep, as opposed to heuristically solving instances of the mixed-objective $p$-IDDMP problem directly. Section 4.3.3 now follows with an in-depth exploration of SP1 for BO-IDDMP, and demonstrates how this problem can be linearized and solved using standard MILP solvers.

### 4.3.3 Design-Only Problem

When applying APP to BO-IDDMP, we noted previously that a natural choice of the heuristic mapping $\mathcal{H}$ is the one that maps any design $\boldsymbol{x}$ to the "fully active" maintenance policy which places any installed components into repair as soon as they become damaged. This mapping is desirable in part because it allows SP1 to find both single-objective optimal solutions of BO-IDDMP: the empty solution for minimizing cost, and the most reliable design being actively maintained for minimizing failure rate. We call this sub-problem the *Bi-Objective Design-Only Problem* (BO-DOP). Not only is BO-DOP useful within the APP framework, but it's also useful in its own right as a point of comparison against BO-IDDMP, as we can compare the solution sets between the two problems. BO-DOP can be considered the static or non-dynamic version of the problem, as typically seen in the repairable RAP literature. The comparison between BO-DOP and BO-IDDMP will be carried out in Section 4.4.

If a component of type $i$ is always repaired as soon as it becomes damaged, then it is healthy with probability $p_i = \tau_i/(\tau_i + \alpha_i)$ and repairing with probability $q_i = 1 - p_i = \alpha_i/(\tau_i + \alpha_i)$. Therefore, in BO-DOP, any copy $j$ of component type $i$ under

design decision $x_{ij}$ is healthy with probability $p_i x_{ij}$, non-healthy (i.e., repairing or non-existent) with probability $1 - p_i x_{ij}$, and repairing with probability $q_i x_{ij}$. The two objectives are therefore as follows:

$$g^o_{\text{DOP}}(\boldsymbol{x}) = \sum_{i=1}^{N}\sum_{j=1}^{M_i} r_i q_i x_{ij} + \sum_{i=1}^{N}\sum_{j=1}^{M_i} c_i p_i x_{ij} \left(\prod_{k<i}\prod_{l=1}^{M_k}(1 - p_k x_{kl})\right)\left(\prod_{l<j}(1 - p_i x_{il})\right) \tag{4.3.1}$$

$$g^f_{\text{DOP}}(\boldsymbol{x}) = \prod_{i=1}^{N}\prod_{j=1}^{M_i}(1 - p_i x_{ij}). \tag{4.3.2}$$

Objective (4.3.1) appears quite complex, so we provide an explanation. The first double-summation states that for every component of type $i$ and copy $j$, we incur repair cost-rate $r_i$ with probability $q_i x_{ij}$. The second summation arises from the fact that for every component type $i$ and copy $j$, we incur from it usage cost $c_i$ with the probability that component $i$ copy $j$ is healthy ($p_i x_{ij}$) and all cheaper components are non-healthy (expressed by the double product) and the first $j-1$ copies of component type $i$ are all non-healthy (expressed by the single product). Objective (4.3.2) is simply the probability that all components are non-healthy. BO-DOP can be expressed as follows:

$$\begin{aligned} \text{(BO-DOP)} \quad & \min_{\boldsymbol{x}} g^o_{\text{DOP}}(\boldsymbol{x}) \\ & \min_{\boldsymbol{x}} g^f_{\text{DOP}}(\boldsymbol{x}) \\ & \text{s.t. } (4.2.8), (4.2.10), (4.2.11), \end{aligned}$$

where we recall Constraints (4.2.8), (4.2.10), (4.2.11) are the knapsack, symmetry breaking and binary constraints, respectively. As currently formulated, BO-DOP is a nonlinear integer program, and would be very difficult to solve exactly. In order to solve the problem, we consider a single-objective, $\varepsilon-$constrained and slightly aug-

mented problem which allows for linearization. We choose the system failure objective to be treated as an $\varepsilon$-constraint. The $\varepsilon-$constraint method is a standard scalarization method for bi-objective problems, and is chosen over the weighted-sum approach to recover a single-objective problem because the weighted-sum approach is known to miss non-supported efficient solutions in the case of multi-objective mixed integer programs (Ehrgott, 2005). Additionally, our preliminary experimentation with the weighted-sum approach to BO-DOP led to numerical instability when compared with the $\varepsilon-$constraint method. Therefore, we focus on the latter method. We augment the problem by adding a small penalty for system failure into the operational cost objective function, using an additional parameter $\delta \geq 0$. As such, we call the resulting problem $\varepsilon - \delta-$DOP.

$$
\begin{aligned}
(\varepsilon-\delta-\text{DOP}) \qquad \min_{\boldsymbol{x}} \; & \sum_{i=1}^{N}\sum_{j=1}^{M_i} r_i q_i x_{ij} \\
& + \sum_{i=1}^{N}\sum_{j=1}^{M_i} c_i p_i x_{ij} \left(\prod_{k<i}\prod_{l=1}^{M_k}(1-p_k x_{kl})\right)\left(\prod_{l<j}(1-p_i x_{il})\right) \\
& + (1+\delta)c_N \prod_{i=1}^{N}\prod_{j=1}^{M_i}(1-p_i x_{ij}) && (4.3.3)\\
\text{s.t. } & (4.2.8), (4.2.10), (4.2.11), && (4.3.4)\\
& \sum_{i=1}^{N}\sum_{j=1}^{M_i} \log(q_i) x_{ij} \leq \varepsilon. && (4.3.5)
\end{aligned}
$$

Objective (4.3.3) is $g^o_{\text{DOP}}$ with an additional penalty term $(1+\delta)c_N$ for system failure. This term ensures that the objective function considers system failure to be at least as bad as using the most expensive component type, which is required for the linearization method employed in the next step. Constraints (4.3.4) are the the knapsack, symmetry breaking and binary constraints used in BO-DOP. Constraint (4.3.5) is the (log) $\varepsilon-$constraint, ensuring that the log failure rate (LFR) falls below some target value $\varepsilon$.

As it stands, $\varepsilon - \delta-$DOP is a heavily nonlinear integer program, and is in fact a

supermodular optimization problem (as shown in Section A.2.1), making it very hard to solve directly. Fortunately, the objective function in $\varepsilon-\delta-$DOP is very similar to that of the well-studied Unreliable p-Median Problem. In relation to this problem, O'Hanley et al. (2013) developed an exact linearization method known as *probability chains*, and we shall apply this method to our problem in order to obtain a formulation that can be solved exactly using standard MILP solvers. The resulting problem, $\varepsilon-\delta-$DOP-PC, is given as follows:

$$
\begin{aligned}
(\varepsilon-\delta-\text{DOP-PC}) \quad & \min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}} \sum_{i=1}^{N}\sum_{j=1}^{M_i}(r_i q_i x_{ij} + c_i y_{ij}) + (1+\delta)c_N z_{N,M_N}, \\
& \text{s.t. } (4.3.4)-(4.3.5) \\
& \quad y_{11} + z_{11} = 1, && (4.3.6) \\
& \quad y_{i(j+1)} + z_{i(j+1)} = z_{ij}, \text{ for all } i = 1,...,N, j = 1,...,M_i - 1, \\
& \quad y_{i1} + z_{i1} = z_{(i-1)M_{i-1}}, \text{ for all } i = 2,...,N, && (4.3.7) \\
& \quad y_{ij} \le p_i x_{ij}, \text{ for all } i = 1,...,N, j = 1,...,M_i && (4.3.8) \\
& \quad y_{i(j+1)} \le p_i z_{ij}, \text{ for all } i = 1,...,N, j = 1,...,M_i - 1 && (4.3.9) \\
& \quad y_{(i+1)1} \le p_{i+1} z_{iM_i}, \text{ for all } i = 1,...,N-1, && (4.3.10) \\
& \quad y_{ij}, z_{ij} \ge 0.
\end{aligned}
$$

For clarity of exposition, we say component $(i,j)$ *precedes* $(i',j')$ if $i < i'$ or $i = i'$ and $j < j'$. The variable $y_{ij}$ represents the probability that component $(i,j)$ is healthy and must be used (i.e., all preceding components are damaged), and the variable $z_{ij}$ is the probability that component $(i,j)$ would be used if it were not damaged, i.e., the probability that component $(i,j)$ and all preceding components are damaged. To achieve this, constraints (4.3.6) - (4.3.7) ensure that $y_{ij}$ and $z_{ij}$ always sum to the $z$ value of the preceding component, constraint (4.3.8) ensures $y_{ij}$ is bounded above by $p_i$ if component $(i,j)$ is installed, or 0 otherwise. Constraints (4.3.9) and (4.3.10) then

ensure that $y_{ij}$ is bounded above by $p_i z_{i,j-1}$ if $j > 1$, or $p_i z_{i-1,M_{i-1}}$ if $j = 1$. The penalty term in front of $z_{N,M_N}$, which outweighs the costs for each $y_{ij}$, ensures that these bounds are attained at optimality if the component is installed. As such, these constraints together ensure that the $y$ and $z$ variables take on the desired values at optimality:

$$y_{ij} = \left(\prod_{k<i}\prod_{l=1}^{M_k}(1-p_k x_{kl})\right)\left(\prod_{l<j}(1-p_i x_{il})\right)p_i x_{ij},$$
$$z_{ij} = \left(\prod_{k<i}\prod_{l=1}^{M_k}(1-p_k x_{kl})\right)\left(\prod_{l\leq j}(1-p_i x_{il})\right).$$

The rigorous proof of equivalence between $\varepsilon-\delta-$DOP-PC and $\varepsilon-\delta-$DOP follows immediately from Proposition 1 of O'Hanley et al. (2013), where the only difference is that our formulation has extra indices for repeated components. As such, the nonlinear integer optimization problem $\varepsilon-\delta-$DOP with $\sum_{i=1}^{N} M_i$ integer variables is transformed into a mixed-integer linear program with the same number of integer variables, and $2\sum_{i=1}^{N} M_i$ continuous variables which take on all of the nonlinearities from the original problem. This results in a much easier problem to solve in general.

Problem $\varepsilon-\delta-$DOP-PC is a MILP that can be easily solved by any MILP solver. We can then re-solve this MILP for different values of $\varepsilon$ to obtain a set of efficient solutions to BO-DOP. To assist the $\varepsilon-$constrained method, it is also worthwhile to consider the two problems that only consider one objective, whilst ignoring the other. These problems give the extreme points of our Pareto front. In the case of minimizing operational costs, the (trivial) optimal solution is to install no components and incur no costs, hence leaving the system permanently failed. In the case of minimizing failure probability, we call the MILP that minimizes failure probability while neglecting operational costs the

Failure Design-Only Problem (F-DOP), and formulate it as follows:

$$\begin{aligned} \text{(F-DOP)} \quad & \min_{\boldsymbol{x}} \sum_{i=1}^{N} \log(q_i) x_i \\ & \text{s.t. } A\boldsymbol{x} \leq \boldsymbol{b} \\ & \quad x_i \in \mathbb{Z}_{\geq 0} \text{ for } i = 1, ..., N. \end{aligned}$$

This allows us to obtain the solution to the bi-objective problem that gives all preference towards the objective of system reliability, and also provides a bound for our $\varepsilon$ values. The models $\varepsilon - \delta-$DOP-PC and F-DOP can both be used to provide solutions to BO-DOP.

### 4.3.4 Algorithm Outline

We now give a brief overview and illustration of the algorithmic implementation of APP when applied to BO-IDDMP. See Section A.2.2 for the full algorithms and detailed explanations. We start by solving BO-DOP, which is SP1 for BO-IDDMP. We do this by solving F-DOP using a MILP solver to obtain the most reliable solution, and otherwise use the $\varepsilon-$constraint method to obtain the other solutions. For each value of $\varepsilon$ we wish to test, a solution is obtained via a MILP solver on $\varepsilon - \delta-$DOP-PC. Our implementation of the $\varepsilon-$constraint method uses two parameters: the starting value $\varepsilon_{\min}$, and the increment value $\Delta\varepsilon$ (note that both of these are negative, as $\varepsilon$ is a log-probability). After solving the problem using $\varepsilon_{\min}$, the solution obtained will have a value for $\ln g^f$, the log failure rate (LFR) of the solution. We update the target value $\varepsilon$ for the next solution by setting $\varepsilon = \ln g^f + \Delta\varepsilon$. This is done iteratively for each new solution, until $\varepsilon$ would exceed the minimum value of LFR found by F-DOP.

Next, we solve SP2($\boldsymbol{x}$) for each design found by BO-DOP. SP2($\boldsymbol{x}$) is simply the dynamic maintenance problem BO-DMP as outlined in Section 4.2.1. This is solved using the dichotomic method of Aneja and Nair (1979) to choose the weights of the

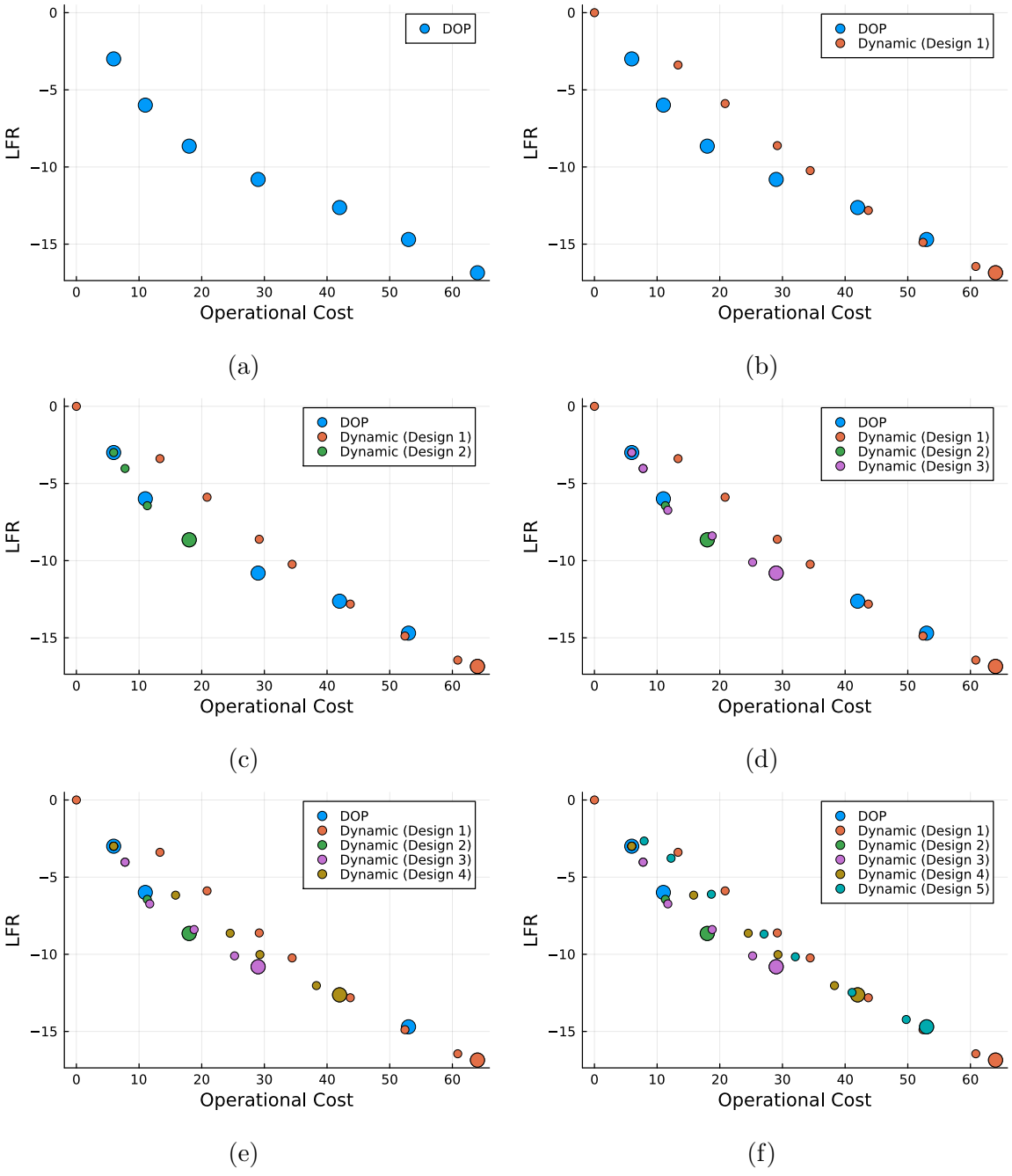



Figure 4.3.1: Illustration of APP building up the population of candidate solutions.

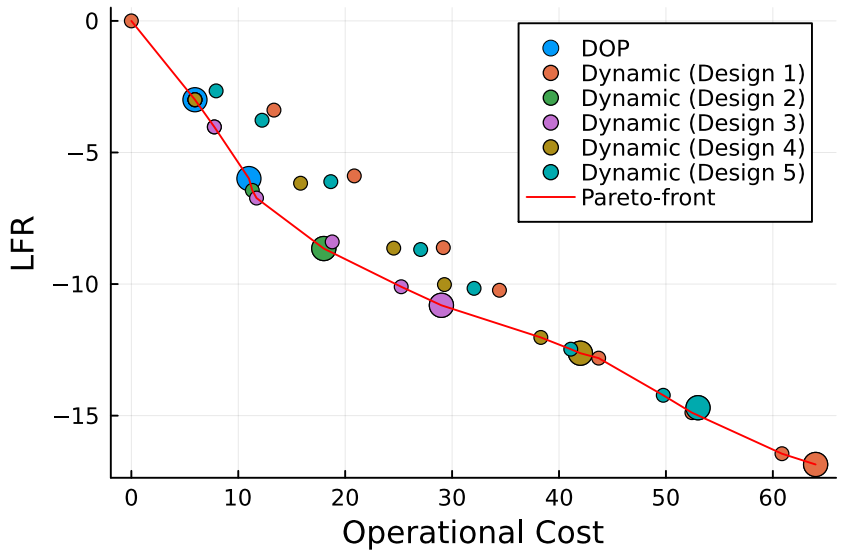



Figure 4.3.2: Pareto front of candidate population.

two objective, which are then normalized to give a weight of 1 the first objective, and a value of $p$ for the $p$-DMP formulation.

The solutions found by SP2 all form a population of candidate solutions. The final step is to identify all non-dominated solutions in our population, and we return this as our result. To provide further intuition for our methodology, Figure 4.3.1 illustrates how APP builds up the population of candidate solutions. Figure 4.3.1(a) shows the set of static solutions that come from solving BO-DOP. Each of the subsequent subfigures then selects a design in a unique color, and adds in the Pareto front of dynamic policies using that design in the same color. Figure 4.3.2 shows the final population, with the Pareto front of this population shown by the red line. Albeit subtle (in this example), this front shows how a lazy dynamic maintenance policy from the orange (right-most) design dominates the teal (second-right-most) design under the fully active policy. This demonstrates the benefit of considering dynamic policies on more reliable designs, as opposed to just the design-only problem with fully active policies. Additionally, we see that two policies from the teal design belong to the non-dominated front, despite the base design being dominated. This shows that we should not discard designs from SP1

just because that solution is dominated by a solution from SP2, because that design may still be useful elsewhere in the front. With our heuristic fully described, Section 4.4 now follows with a computational study comparing the exact and heuristic approaches, and investigates the impact of problem parameters.

## 4.4 Numerical Examples and Computational Results

In this section we present the results of computational experiments in order to evaluate the performance of the APP heuristic, and also explore the effects of varying the parameters in our model. Section 4.4.1 described how we adapt a well known test instance for series-parallel models into a suite of test instances for our parallel model. In Section 4.4.2 we show that in general our heuristic finds better populated Pareto fronts than the exact method without sacrificing the quality of the solutions found, and does so in far less time than the exact method. We do this by adapting a popular problem set from the RAP literature. In Section 4.4.3 and Section 4.4.4 we investigate the effects of varying usage costs and repair costs on the solutions to our problem in order to demonstrate the importance and impact of these parameters in our model. We demonstrate that such variations have an impact on dynamic policies. The code used can be found on GitHub.

### 4.4.1 Test Instances

We use 14 different sets of components based on test instances found in the RAP literature, following the installation cost, weight, and reliability values of the 14 subsystems introduced by Fyffe et al. (1968). To be clear, whereas the problem in Fyffe et al. (1968) is a single series-parallel problem with 14 subsystems, we treat each subsystem as a separate one-subsystem parallel problem to obtain a set of parallel-only problems.

The original problem set defines, for each component $i$, its reliability $p_i$, its installation cost $C_i$, and its weight $w_i$. For the parameters unique to our model, for each component $i$ we set $\tau_i = 1$ and obtain $\alpha_i$ by solving $\tau_i/(\tau_i + \alpha_i) = p_i$. We set usage cost $c_i = 1$ and repair cost $r_i = 100$ for the initial problem set. The impacts of varying these parameters will be explored in subsequent subsections. For each set we use different constraint values $b$, which are in turn applied to both budget and weight, obtaining knapsack constraints $\boldsymbol{C}^T x \leq b$ and $\boldsymbol{w}^T x \leq b$. Clearly, as the budget $b$ increases, the number of possible copies of each component increases, and therefore the size of the state space in the MILP increases. For each component set, we choose values of $b$ corresponding to the third up to the eighth multiples of the minimum weight found in the component set. For example, if the minimum weight in a set is 5, we test budgets of 15, 20, 25 and so on up to and including 40, reflecting problem instances that allow from 3 to 8 copies of the lightest component, respectively. We denote problem instances by $(a, b)$, where $a$ is the component set (i.e., subsystem from the original problem), and $b$ is the budget. For example, instance $(5, 15)$ includes components from the fifth subsystem and has a budget of 15.

### 4.4.2 Runtime and Scalability

In this section we solve the BO-IDDMP using an exact method, the heuristic method described in Section 4.3, and the non-dynamic BO-DOP via probability chains. For each problem instance, we compare the solutions with respect to runtimes and the numbers of solutions found. All MILPs and LPs are solved using Gurobi (Gurobi Optimization, LLC, 2024) and our own fork of MultiObjectiveAlgorithms.jl (Dowson et al., 2025), and all code is written in Julia 1.9.3. The code is run on a computer with an Intel Xeon Gold 6348 with four cores clocked at 2.6 GHz and 16 GB of RAM.

For each problem instance as defined in Section 4.4.1, we solve the problem in three ways: (i) using an exact method by applying the dichotomic method to the BO-IDDMP

formulation and solving the resulting MILPs using Gurobi; (ii) using the APP heuristic, with parameters $\varepsilon_{\min} = 0$, $\Delta\varepsilon = -0.1$, and $\delta = 0.1$; (iii) using the probability chains method to solve BO-DOP, using the same parameters as used in APP. We set a soft time limit of 3600 seconds to both build the model and solve it for the prescribed range of penalty values, but with flexibility to allow Gurobi and the MultiObjectiveAlgorithms package to terminate "gracefully" when this limit is exceeded.

We defer the full two-page table of results to Section A.3.1, and discuss the key findings here. In general, we found that our implementation of APP for this problem performed very well in terms of runtimes and completeness of the Pareto front. For some of the smaller instances (i.e., those with smaller budgets and therefore easily solved MILPs) we found that the exact method was faster than the APP heuristic, but for the majority of instances APP was much faster. Exact solution times ranged from 0.31 to 3410.58 seconds for times within the time limit, with four instances exceeding this due to the soft time limit: instance $(6, 28)$ with 4624.9 seconds, $(6, 32)$ with 7196.05 seconds (failing to find even one solution), $(9, 56)$ with 7196.93 seconds (again failing to find a solution), and $(14, 48)$ with 5974.65 seconds. An inspection of the Gurobi logs revealed that these instances encountered difficult linear relaxations. On the other hand, APP solution times ranged between 0.32 and 6.30 seconds. Generally, for any fixed set of components, the solution time grows drastically for the exact solver as the budget increases, and grows far more slowly for the APP method. This is because designs identified by the APP method generally only have a small number of component types (usually two), meaning that the MDP associated with any given design to be solved in SP2 is quite small. This contrasts with the exact method, which needs to enumerate all possible state-action pairs across all designs.

Both solution methods find a large number of solutions for problems with larger budgets, despite a visual inspection of the Pareto front suggesting less diversity in the solution space. This is because two distinct Pareto optimal policies may only

differ in the actions taken in states that occur with very low probability, meaning that the overall objective value is very similar yet not exactly the same. Instance (14,42) provides an example of this: the exact solution method finds 34 solutions and APP finds 29 solutions; however, a visual inspection of Figure 4.4.1 would only suggest a total of about 17 distinct solutions across both methods. Note that all orange-colored APP points overlap a blue-colored exact point, and that the one exact solution not found by APP is still very close to an APP solution. To investigate the ability of APP to find the same or very similar solutions to the exact method, we plotted the Pareto fronts of all instances where at least one of the exact Pareto-optimal solutions did not have an APP solution within 2% of it in terms of both log failure rate (LFR) and operational costs. The majority of these plots proved uninteresting and can be found in the Section A.3.2, as all but two instances found that all exact solutions had a nearby APP solution upon visual inspection, or showed that the missed exact solution was itself dominated due to numerical error (this started to occur when failure probabilities dropped below $10^{-9}$). The two exceptions were instances (5,12) and (5,24). Instance (5,12) is the only instance with very obvious solutions missed by APP but found by the exact method, and also showcases that one of the solutions found by APP is dominated by an exact solution, as shown in Figure 4.4.2. In this case, one of the designs used by the exact method (including the one that dominates an APP solution) uses 1 copy of each component, whereas this design is not found at all by APP. As such, this is a rare case where there is a design that is not Pareto-optimal for BO-DOP, but which lends itself well to dynamic maintenance. As rare as this instance is, it showcases a flaw of the APP method, in that it does not explicitly search for solutions that work well only under exact dynamic policies. Of course, such a thing is hard to quantify, but could indicate an interesting future research direction. A similar but less dramatic example of domination is found in instance (5,24), where the exact method finds a design with 3 copies of the second and third component type, and this design is not found by APP.

The experiment yielded 18 out of 84 instances where at least one DOP solution was dominated by an IDDMP solution. This means that in these instances, there exist design-policy pairs with dynamic policies that outperform an "optimal" design with a fully active maintenance policy across both objectives. This suggests that, in some cases, it is both cheaper and more reliable to use a dynamic maintenance policy on a design with more inherent reliability than it is to design a system to strike a specific balance between the two objectives and maintain all components actively. This generally occurs for instances with larger budgets, and therefore more capacity to install a large number of redundant components, giving more flexibility to the dynamic maintenance policies. Figure 4.4.3 provides the Pareto front of Instance (2,64), in which 4 of the 9 design-only solutions are dominated by at least one IDDMP solution. Details of all problem instances can be found in the Section A.3.1.

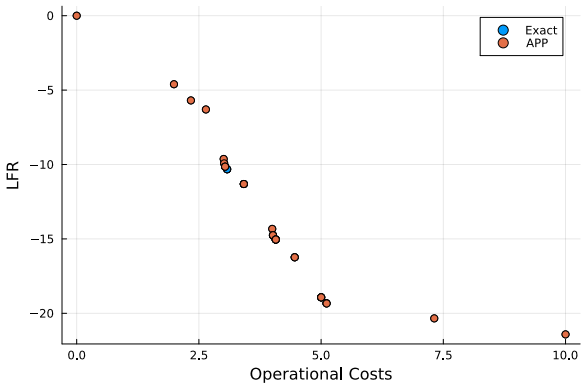



Figure 4.4.1: Pareto front for instance (14,42)

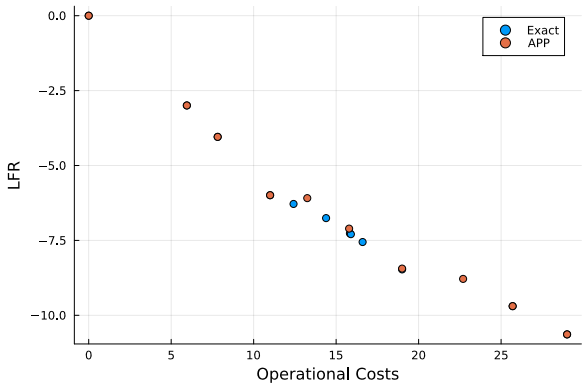



Figure 4.4.2: Pareto front for instance (5,12)

### 4.4.3 Effect of Heterogeneous Usage Costs

In our model we consider the inclusion of usage costs, corresponding to the costs of using components when there are no cheaper components in healthy condition. Usage costs have not typically been included in previous RAP formulations, and therefore it is useful to investigate their effects on the resulting optimal solutions. In this subsection

| $i$ | $p$ | $\tau$ | $c$ | $r$ | $C$ | $w$ |
|---|---|---|---|---|---|---|
| 1 | 0.99 | 1 | 1 | 100 | 3 | 5 |
| 2 | 0.98 | 1 | 1 | 100 | 3 | 4 |
| 3 | 0.97 | 1 | 1 | 100 | 2 | 5 |
| 4 | 0.96 | 1 | 1 | 100 | 2 | 4 |
| Budget | - | - | - | - | 20 | 20 |

Table 4.4.1: Base problem parameters.

| $c_1$ | $c_2$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $g^o$ | $\ln g^f$ |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 1.99 | -4.61 |
| | | 2 | 0 | 0 | 0 | 3.00 | -9.21 |
| | | 3 | 0 | 0 | 0 | 4.00 | -13.82 |
| | | 4 | 0 | 0 | 0 | 5.00 | -18.41 |
| 10 | 1 | 0 | 1 | 0 | 0 | 2.98 | -3.91 |
| | | 1 | 1 | 0 | 0 | 4.18 | -8.52 |
| | | 2 | 1 | 0 | 0 | 5.18 | -13.12 |
| | | 3 | 1 | 0 | 0 | 6.18 | -17.73 |
| 10 | 10 | 0 | 0 | 1 | 0 | 3.97 | -3.51 |
| | | 1 | 0 | 1 | 0 | 5.27 | -8.11 |
| | | 2 | 0 | 1 | 0 | 6.27 | -12.71 |
| | | 3 | 0 | 1 | 0 | 7.27 | -17.32 |
| | | 0 | 4 | 0 | 1 | 13.36 | -18.87 |
| 100 | 100 | 0 | 0 | 1 | 0 | 3.97 | -3.51 |
| | | 0 | 0 | 2 | 0 | 7.00 | -7.01 |
| | | 1 | 0 | 1 | 0 | 7.94 | -8.11 |
| | | 1 | 0 | 2 | 0 | 8.09 | -11.62 |
| | | 2 | 0 | 1 | 0 | 8.97 | -12.72 |
| | | 2 | 0 | 2 | 0 | 9.09 | -16.22 |
| | | 3 | 0 | 1 | 0 | 9.97 | -17.32 |
| | | 0 | 3 | 0 | 2 | 15.16 | -18.17 |
| | | 0 | 4 | 0 | 1 | 16.96 | -18.87 |

Table 4.4.2: Effect of usage costs on efficient designs.

| $r_1$ | $r_2$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $g^o$ | $\ln g^f$ |
|---|---|---|---|---|---|---|---|
| 100 | 100 | 1 | 0 | 0 | 0 | 1.99 | -4.61 |
| | | 2 | 0 | 0 | 0 | 3.00 | -9.21 |
| | | 3 | 0 | 0 | 0 | 4.00 | -13.82 |
| | | 4 | 0 | 0 | 0 | 5.00 | -18.42 |
| 300 | 100 | 0 | 1 | 0 | 0 | 2.98 | -3.91 |
| | | 1 | 0 | 0 | 0 | 3.99 | -4.61 |
| | | 0 | 2 | 0 | 0 | 5.00 | -7.82 |
| | | 1 | 1 | 0 | 0 | 6.00 | -8.52 |
| | | 0 | 3 | 0 | 0 | 7.00 | -11.74 |
| | | 1 | 2 | 0 | 0 | 8.00 | -12.43 |
| | | 0 | 4 | 0 | 0 | 9.00 | -15.65 |
| | | 1 | 3 | 0 | 0 | 10.00 | -16.34 |
| 300 | 300 | 1 | 0 | 0 | 0 | 3.99 | -4.61 |
| | | 1 | 0 | 1 | 0 | 6.9997 | -8.11 |
| | | 2 | 0 | 0 | 0 | 7.00 | -9.21 |
| | | 3 | 0 | 0 | 0 | 10.00 | -13.82 |
| | | 4 | 0 | 0 | 0 | 13.00 | -18.42 |
| | | 0 | 4 | 0 | 1 | 29.00 | -18.87 |
| 500 | 500 | 0 | 0 | 1 | 0 | 3.97 | -3.51 |
| | | 1 | 0 | 0 | 0 | 5.99 | -4.61 |
| | | 0 | 0 | 2 | 0 | 7.00 | -7.01 |
| | | 1 | 0 | 1 | 0 | 9.00 | -8.11 |
| | | 0 | 0 | 3 | 0 | 10.00 | -10.52 |
| | | 1 | 0 | 2 | 0 | 12.00 | -11.62 |
| | | 0 | 0 | 4 | 0 | 13.00 | -14.03 |
| | | 1 | 0 | 3 | 0 | 15.00 | -15.12 |
| | | 2 | 0 | 2 | 0 | 17.00 | -16.22 |
| | | 3 | 0 | 1 | 0 | 19.00 | -17.32 |
| | | 4 | 0 | 0 | 0 | 21.00 | -18.42 |
| | | 0 | 4 | 0 | 1 | 45.00 | -18.87 |

Table 4.4.3: Effect of repair costs on efficient designs.

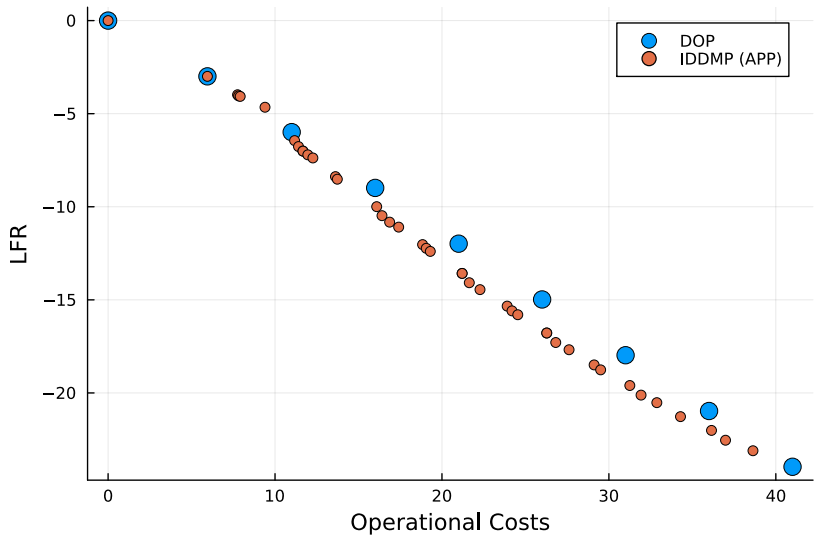



Figure 4.4.3: Pareto front for instance (2,64)

we consider only the static BO-DOP as opposed to the full BO-IDDMP. This is for two main reasons: firstly, the static BO-DOP is sufficiently novel to demonstrate the effects of usage costs without introducing the complications of dynamic maintenance policies, and secondly, the results in Section 4.4.2 have already shown that designs found in exact solutions to BO-IDDMP but not found by BO-DOP are somewhat rare.

Throughout this and subsequent subsections, we focus on problem 6-20 from the problem set given in Section 4.4.2. The base parameters for this problem are given in Table 4.4.1. In this base problem, we see that components 1 and 2 dominate components 3 and 4 respectively in everything except installation costs $C$, which are a weaker constraining factor than weight. As such, we focus on varying $c_1$ and $c_2$, the usage costs of components 1 and 2. We consider different combinations of the cost values 1, 10, and 100. The pair of costs $c_1$ and $c_2$, the design solutions $\boldsymbol{x}$ in general integer form, and the respective two objectives of operational costs $g^o$ and log failure rate $\ln g^f$ are reported in Table 4.4.2. Omitted from the table for each problem is the solution $\boldsymbol{x} = (0, 5, 0, 0)$, which is the solution to F-DOP, and is therefore always included in the front.

Cost combination $(c_1, c_2) = (1, 1)$ has Pareto-optimal solutions which are all multi-

ples of component 1 (excluding the F-DOP solution), which makes sense intuitively as this is the most reliable component. This not only helps in the reliability objective, but also decreases long-run repair costs. For costs $(10, 1)$, we see that all solutions excluding F-DOP instead have at least one copy of component 2, and some number of copies (including none) of component 1. This also aligns with our intuition, since component 1 is still the most reliable and the least expensive to maintain. Additionally, just one copy of component 2 means that we do not have to worry about the higher cost-rate of component 1 most of the time, so the effect of this higher value becomes negligible.

For cost combination $(10, 10)$, we see that components 3 and 4 start to become more desirable. The first four solutions are of a similar type to those found for $(10, 1)$, as they all have one copy of a component with a low usage cost - which offsets the long-run impact of all other usage costs in the solution - and then some number of copies of component 1. We also have the solution $x = (0, 4, 0, 1)$. This solution sees an interplay between the two objectives and the weight budget. We would like to add an extra component to $(3, 0, 1, 0)$ for further reliability, but this would exceed our weight budget, so we instead have to use the lighter components to achieve a similar solution. This solution still follows the same pattern of having one component with a low usage cost and a number of copies of a higher reliability component.

The final combination $(100, 100)$ is a very extreme example, because in this case the usage costs and repair costs for components 1 and 2 are equal. Despite this, they do not go unused, because they can be used alongside the cheaper-to-use components 3 and 4, so the extreme usage cost has a low impact. We now see solutions such as $(1, 0, 2, 0)$ and $(2, 0, 2, 0)$, which include two copies of the cheaper component as opposed to the one copy we saw for previous problems. The interpretation of this is not quite as clear, but it could be due to the fact that multiple copies of a cheap-to-use component further offset the contribution to the long-run average cost of the expensive-to-use yet reliable component 1. An explainable managerial insight here is that, aside from the choice of

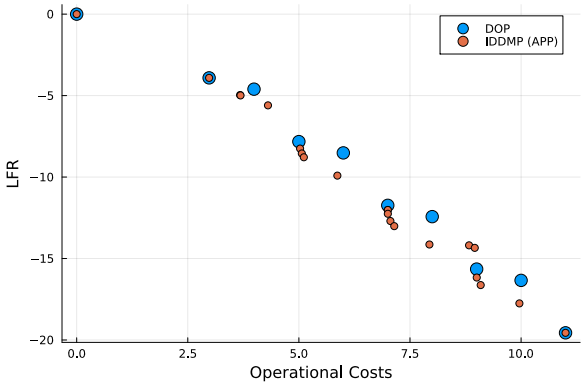



Figure 4.4.4: Pareto fronts of DOP and IDDMP solutions for $(r_1, r_2) = (300, 100)$.

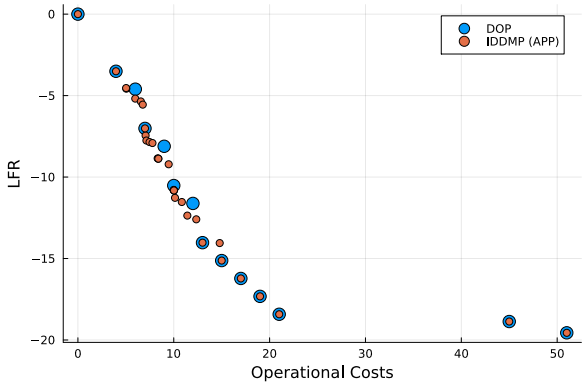



Figure 4.4.5: Pareto fronts of DOP and IDDMP solutions for $(r_1, r_2) = (500, 500)$.

some "primary" cheap-to-use component type, the usage costs of backup components may be seen as a negligible consideration outside of extreme cases.

### 4.4.4 Effect of Heterogeneous Repair Costs

In addition to usage costs, another aspect of our model that differentiates it from much of the existing RAP literature is the inclusion of heterogeneous repair costs, whereby different components have different repair costs per unit time. We are aware of only one previous study that makes similar assumptions about repair costs (Lins and Droguett, 2011), although several other works consider repair costs attributable to repair workers being assigned to subsystems. In this subsection we explore the impact of these costs on solutions to the problem, again using instance 6-20 from Section 4.4.2 as a basis. We again vary the repair cost-rate parameters of components 1 and 2, $r_1$ and $r_2$. Table 4.4.3 reports the repair costs being investigated, their Pareto-optimal BO-DOP solutions, and the objective values of these solutions under the fully active repair policy.

The case $(r_1, r_2) = (100, 100)$ is simply a re-run of the base problem. Case $(300, 100)$ demonstrates the effect of tripling the repair cost-rate of the first component. Immediately of note is the fact that we obtain more design solutions. These can be split into two groups: those with one copy of component 1, and those with no copies of compo-

nent 1. In the DOP case this is not particularly notable, but in the IDDMP case we see that all designs that use a copy of component 1 are dominated by a design without this component, when used alongside a dynamic maintenance policy. This demonstrates another situation where dynamic maintenance policies not only fill in the Pareto front, but actually dominate some of the static solutions. This is evident from the plot of the Pareto front in Figure 4.4.4. Notably, this situation is not unrealistic, as it could be quite reasonable that a more reliable component is more expensive to perfectly repair or replace when it does fail.

Case $(300, 300)$ still primarily shows preference towards the first component, similarly to the base problem. Particularly of note within this problem is that it highlights an idiosyncrasy of our model. Solutions $(1, 0, 1, 0)$ and $(2, 0, 0, 0)$ are very similar. Their operational costs due to repairs are identical, as $r_1 q_1 = 300(1 - 0.99) = 3$ and $r_3 q_3 = 100(1 - 0.97) = 3$, and the usage costs of all components are identical. However, design $(1, 0, 1, 0)$ is slightly less reliable than $(2, 0, 0, 0)$, and therefore it incurs usage costs slightly less often and is not dominated by the other solution. In reality, it is clear that design $(1, 0, 1, 0)$ should not be chosen as its usage costs are almost identical to those of $(2, 0, 0, 0)$, and the latter solution is more reliable.

Case $(500, 500)$ has a very populated Pareto front for the BO-DOP, with none of these solutions being idiosyncratic as described in the previous case, and with only two of these designs, namely $(1, 0, 1, 0)$ and $(1, 0, 2, 0)$, being dominated by a dynamic solution (see Figure 4.4.5). The rest of the solutions are reasonably well spaced in terms of both objectives, with large jumps in cost only occurring between $(4, 0, 0, 0)$ and $(0, 4, 0, 1)$, and from that solution to $(0, 5, 0, 0)$, which has a cost of 51. We observe that these solutions are quite diverse in their choices of component, with component 3 now becoming more preferable across the solution set.

### 4.4.5 Managerial Insights

We can briefly summarize some of our findings at the qualitative level, in a way that could provide applicable advice for redundant systems more complex than those explored here. The first thing to note, which is apparent from Table 4.4.2 and Table 4.4.3 and which was common across all experiments, is that the optimal system designs generally only included two types of component. We can interpret this as including a "primary" component that is better in terms of usage costs, and a "secondary" component which is better in terms of reliability and maintenance costs. At an intuitive level, it makes sense that this would hold for many systems with redundancy, where one type of component is included because it has a good balance between all factors including usage costs (where this cheaper usage cost is the one that will be incurred most of the time), and another type will be included for reasons other than its usage cost (i.e., where we don't mind occasionally paying this higher usage cost, because this won't happen very often). Of course, it may sometimes be the case that the optimal design of a system includes more types of component (in the case of awkward knapsack constraints, for example), but this two-component-type design principle seems reasonable as a baseline.

The second aspect to summarize is the instances where dynamic policies seem to work the best. Across all instances tested in Section 4.4.2, we found that dynamic policies always provided a richer and more complete set of Pareto-optimal solutions than the design-only problem, which is restricted to the "always repair" maintenance policy. This shows that dynamic policies can "interpolate" between designs in terms of their reliability and operational costs, allowing decision-makers to strike a more specific balance between the two. This is particularly useful when there are large gaps in the Pareto front of the design-only problem. We also found a number of instances where a dynamic policy dominated a design-only solution, and noted that this only occurred in problem instances with a larger budget, and therefore more scope for extra redundancy.

A key takeaway here is that dynamic maintenance policies are at their best (i.e., they can dominate static policies) for systems with lots of redundant components, and this is likely due to the inherent leeway granted by such systems. This is best demonstrated by Figure 4.4.3, which shows many static solutions dominated by dynamic solutions. Another insight is that the dynamic solutions show a more sub-linear relationship between costs and LFR compared to the mostly linear relationship between the two for static solutions. This indicates that dynamic solutions can obtain cost savings with a more favorable, less pronounced decrease in reliability. Again, this seems more likely for systems with more redundant components. In real terms, this suggests that it is potentially worthwhile to install additional redundancy into a system and maintain it using a dynamic maintenance policy, especially in situations with more relaxed physical constraints, and where long-run costs and reliability are of greater concern than upfront costs. This additional redundancy could also be useful in situations where repairs for some reason cannot be performed straight away, for example due to repair worker availability or budgetary issues.

While these managerial insights seem to make sense at the high level, it is important to note that we have only discussed parallel systems under simplifying assumptions such as hot standby, perfect switching, and exponential failure and repair times. If this model is used to obtain system designs and dynamic maintenance policies, additional work must be done to "sanity check" the suggested solutions with respect to the real-world system, and to verify the performance of a design and/or maintenance strategy on a model with more of the important real-world details, e.g. by using simulation.

In this section we have demonstrated that our heuristic is capable of providing a multitude of high-quality solutions to the BO-IDDMP in far less time than the exact method, and have showcased and interpreted the effects of usage and repair costs on particular solutions. We have also found that dynamic solutions greatly outnumber static solutions, and in some cases dominate static solutions. This demonstrates that

the integration of strategic design decisions with operational maintenance decisions in a unified model is a worthwhile endeavor.

## 4.5 Conclusion

In this chapter we have introduced a novel BO-MDP Design extension of the RAP, with parameters such as usage and repair costs which are seldom seen in the RAP literature. Due to the complexity of the resulting problem, we constructed a bespoke heuristic to obtain an approximate Pareto front. We compared the performance of this heuristic to that of an exact method for small problem instances inspired by earlier research, and found that our heuristic was much faster than the exact method and also found more solutions due to its reduced sensitivity to parameter values, and that the solutions found were optimal in the majority of cases. We also demonstrated the advantages of considering dynamic maintenance policies, showing that they lead to better populated Pareto fronts, and sometimes find design-policy pairs that dominate static solutions. We also investigated the effects of varying the usage costs and repair costs, and in all cases were able to provide intuitive interpretations of the results.

While our focus on parallel systems is arguably quite restrictive, we believe that this work offers an important first step in integrating system design and dynamic maintenance decisions. Extending to the series-parallel case is non-trivial, as the number of states in the MDP grows exponentially with the number of subsystems, even before adding in design decisions. As such, extending the work presented in this chapter to the series-parallel case offers an interesting direction for future research. We believe that such systems could be decomposed into parallel system subproblems via methods such as Dantzig-Wolfe decomposition, allowing us to use the methods in this chapter to solve those subproblems. There are also many other variations of the RAP as identified in Section 4.1.1, such as $k$-out-of-$n$ systems or complex systems, cold or mixed standby

with imperfect switching, the allocation of repair teams, and multi-state components. A limitation of the model employed in this chapter is the need to assume exponential failure and repair times in order to use an MDP formulation. Further work could be done to evaluate the performances of solutions to our model on more realistic systems with non-exponential event times, or one could go further and integrate general time distributions into the optimization model directly. Another limitation of our model is the hot-standby assumption, where components always have the same constant rate of failure, regardless of whether or not they are being used. The MDP model could be extended to the cold- or warm-standby cases by including switching decisions whenever the in-use component fails, or whenever a component comes back online. Switching decisions would aim to find a balance between usage costs and the reliability and future maintenance costs of available components. On a final note, further work could be done to investigate how to improve upon the heuristic presented in this chapter, or to construct an efficient algorithm for finding exact solutions.

From the methodological point of view, we have contributed to the emerging MDP Design problem. This is a very young and sparse area of literature, and developing the theory of this field should be important and fruitful, due to the potential of applying the model to other real-world problems such as inventory and queueing problems (Brown et al., 2024). We have also introduced the APP method, where for bi-objective MDP Design problems we first obtain a set of first-stage decisions corresponding to a Pareto front based on simplifying assumptions on the second-stage decisions, and then obtain a new Pareto front for each of these first-stage decisions by optimizing the second-stage decisions. It remains to see how well this methodology works for other MDP Design problems.

As we have presented APP as a general heuristic framework for bi-objective MDP Design problems, an interesting research direction would explore how APP could work in other settings. For example, one could consider a queueing design and control problem.

The design aspects could include the hiring of servers with different service rates or the allocation of limited waiting space to different queues, and the second-stage decisions could include the routing of customers to queues. Objectives could be based on server hiring costs, overall customer satisfaction (e.g. due to waiting times), satisfaction of different customer types, etc. A parameterized family of heuristics could be based on static stochastic routing policies, i.e., state-agnostic random routing of customers to queues. These are well-known to provide good baseline policies for queueing systems, which can be improved upon by dynamic polices (Shone et al., 2020). The design problem would then find different designs that balance between the different objectives, and the second stage would find deterministic dynamic policies over each of the designs. Queueing control itself is a hard problem; however, dynamic heuristics can be used that are known to improve upon static stochastic policies (Krishnan, 1990; Argon et al., 2009; Shone et al., 2020).

While the APP method has been successful in our study, we acknowledge that in general APP has no iterative element, and cannot explore solutions beyond its two stages. As such, the development of metaheuristic methods for MDP Design problems remains an interesting research direction.

# Chapter 5

# Decomposable Impulsive Continuous-Time Markov Decision Processes for Maintaining Series-Parallel Systems

## 5.1 Introduction

In this chapter we focus on continuous-time MDPs (CTMDPs) and consider variations of CTMDPs that are seldom seen together in the academic literature despite being commonly present in real-life situations, including CTMDPs with impulsive actions, multi-objective CTMDPs, and decomposable CTMDPs. By developing the necessary modelling frameworks and a heuristic solution methodology, we then investigate a CTMDP model for the dynamic maintenance of series-parallel systems, extending from the simpler parallel systems seen in the previous chapter. For now, we omit the decision stage of such a problem, and focus purely on maintenance. Chapter 6 will then consider the simultaneous design and maintenance optimisation of such systems.

### 5.1.1 Impulsive Actions

One variation of the CTMDP involves the use of *impulsive actions*. These are actions that cause an instantaneous change in the state, as opposed to the more common *gradual* actions which mandate a random non-zero sojourn time before the transition takes place. Such a modelling feature is useful for any problem where the state changes immediately in response to the action. A pioneering theoretical treatment of impulsive control or gradual-impulsive control (for systems that allow for both types of action) is given by Dufour and Piunovskiy (2015). However, we believe this framework to be mathematically complex in such a way as to become unapproachable to some modellers, leaving room for the development of formulations which capture similar behaviour whilst being easier to understand for use in the modelling of real-world problems.

### 5.1.2 Multiple Objectives

The second variation of the MDP we consider is the multi-objective MDP, or MOMDP. In a standard single-objective MDP, one aims to control the system in such a way that the long-run scalar costs are minimised. In a MOMDP, one aims to strike a balance between elements of a vector cost function in the long run, and to find multiple ways to control the system to strike different balances. These different costs can be diametrically opposed, so it is necessary to find a range of compromises. A survey on MOMDPs is given by Roijers et al. (2013). The literature on multi-objective MDPs is relatively sparse; some investigations of such problems include Wakuta (2000) and Ghosh (1990). We argue that many common problems which are modelled as MDPs lend themselves to multi-objective modelling and should be treated as such. To give a few examples: in maintenance problems, natural objectives can be based on maintenance costs and system reliability or performance; in inventory problems, objectives can include holding cost minimisation and demand satisfaction; and in queueing control, objectives can be based on costs and customer satisfaction, where the latter may depend, for example,

on the time spent in the queue. These problems, and many more, require a balance between monetary reward and some measurable outcome which may not scale linearly with — or could be entirely separate from — the monetary outcome.

### 5.1.3 Decomposability

The third variation we consider is that of decomposability, meaning an MDP that can be thought of as being comprised of multiple smaller MDPs, usually with an objective function or constraints linking them together. A common example is that of the weakly-coupled MDP, which is made up of multiple smaller MDPs with their own state spaces, action spaces, and stochastic dynamics, but with global constraints on action sets. For example, costs may be associated with actions taken in each small MDP, and there may be a global budget over all the MDPs for each decision epoch. Explorations of weakly-coupled MDPs are given by Meuleau et al. (1998). Bertsimas and Mišić (2016) introduces a broader definition of a decomposable discrete-time MDP which includes weakly-coupled MDPs, but still limits this to discrete time. In our work, the decomposed MDPs will be linked together by objective functions which may be non-linear, rather than constraints.

### 5.1.4 Chapter Outline

We establish the mathematical framework necessary to extend the MDP model of Chapter 4 to the series-parallel case, and develop a solution methodology for this type of model. Section 5.2.1 and Section 5.2.2 first generalise, formalise, and analyse the implementation of impulsive control (e.g. instant transitions) that was applied in Chapter 4. Notably, we use equivalence relations to analyse this framework, and prove that a particular class of intuitive policies always contain an optimal solution. We also demonstrate that it is always possible to obtain such a policy from an optimal policy not in this class, if one exists. Section 5.2.3 follows by extending this impulsive CTMDP

model to the *decomposable* impulsive CTMDP, a model which can be thought of as multiple separable impulsive CTMDPs which can be controlled together, and which are unified by shared objective functions. Further analysis shows that, when controlled by a specific type of policy, the resulting Markov process decomposes into independent Markov processes for each compartment, up to an equivalence relation. Section 5.3 then presents a block-diagonalisation of the LP formulation of a decomposable impulsive CTMDP under certain restrictions, where this block-diagonal form can then by solved via Dantzig-Wolfe decomposition. This in turn allows us to develop a heuristic in which we find Pareto fronts for the compartment CTMDPs that made up the decomposable impulsive CTMDP, and then piece these solutions back together with a bi-objective binary program. In Section 5.4 we apply our modelling framework and solution methodology to a dynamic maintenance problem motivated by reliability engineering, and investigate the strengths and weaknesses of our solution methodology compared to an exact LP solver. Section 5.5 then concludes with a discussion of strengths and limitations, and with suggestions for further research. Most of the mathematical proofs are deferred to the appendices.

### 5.1.5 Our Contributions

Our first contribution is a method for implementing impulsive control in CTMDPs in an approachable way. Specifically, we focus on finite state and action spaces and therefore avoid the need for technical considerations such as Borel spaces and measurability. Our modelling framework for impulsive control fits within the standard CTMDP framework, and as such it is compatible with standard CTMDP solution methodologies such as uniformisation, value/policy iteration, linear programming and reinforcement learning.

Our second contribution is to extend our modelling framework for impulsive control to include multiple objectives and decomposability. Specifically, we introduce a *decomposable impulsive CTMDP*, which can be thought of as a CTMDP composed of

multiple separate impulsive CTMDPs. We discuss the subtleties of this and develop the necessary structure to ensure that whole CTMDP and smaller compartment CTMDPs are both truly impulsive.

Our third contribution is a heuristic solution methodology for multi-objective decomposable impulsive CTMDPs. We prove that a restricted version of the problem of finding the Pareto front for a decomposable impulsive CTMDP can be transformed to have a block-diagonal structure, which enables the use of Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960). In short, we show that with some additional assumptions on the solution space, finding the Pareto front of the decomposable impulsive CTMDPs is equivalent to finding the Pareto fronts of the compartment CTMDPs and piecing these solutions back together.

### 5.1.6 Preliminaries

This chapter assumes familiarity with continuous-time Markov decision processes, multi-objective optimisation, and Dantzig-Wolfe decomposition. Additionally, the theoretical work makes heavy use of equivalence relations. A brief summary of equivalence relations can be found in Section B.1.

## 5.2 Modelling Framework

This section introduces a succession of modelling frameworks which build upon each other. In Section 5.2.1 we introduce a notion of CTMDPs that allows for dummy "self-transitions". This allows for the development of a range of equivalence classes that will later be useful for the analysis of our impulsive and decomposable frameworks. Section 5.2.2 introduces our formulation of CTMDPs with impulsive control, building upon the CTMDPs with self-transitions. Section 5.2.3 then extends this to decomposable impulsive CTMDPs, formalising the notion of multiple stochastically in-

dependent impulsive CTMDPs being treated as one impulsive CTMDP which share objective functions. Each of Sections 5.2.1, 5.2.2 and 5.2.3 are further subdivided into a "Model" subsection to give a softer introduction, followed by a "Theory" subsection to provide a more rigourous analysis. As such, the reader who is interested only in the modelling frameworks, their interpretations, and how to use them can just read the first part of each subsection.

### 5.2.1 CTMDPs with Self-Transitions

#### Model

Normally, we define a finite-state and finite-action CTMDP as a 4-tuple:

$$\mathfrak{C} = \left\langle \mathcal{S}, (\mathcal{A}(s))_{s\in\mathcal{S}}, q : \mathcal{SA} \times \mathcal{S} \to \mathbb{R}, \boldsymbol{c} : \mathcal{SA} \to \mathbb{R}^k \right\rangle,$$

where $\mathcal{S}$ is the state space, $\mathcal{A}(s)$ is the feasible action space for state $s \in \mathcal{S}$, $\mathcal{SA} = \{(s,a) : s \in \mathcal{S}, a \in \mathcal{A}(s)\}$ is the set of feasible state-action pairs, $q(s,a,s')$ is the transition rate from state $s$ to state $s'$ when taking action $a$, and $\boldsymbol{c}(s,a)$ is the vector cost rate when taking action $a$ in state $s$. We write $\mathcal{A} := \bigcup_{s\in\mathcal{S}} \mathcal{A}(s)$ as the space of all possible actions. We usually consider CTMDPs to be controlled by a *deterministic stationary* (DS) policy, which is a map $\mu : \mathcal{S} \to \mathcal{A}$. Throughout, we assume that our CTMDPs are *weakly-communicating*, meaning the state space can be split into two sets: one for which for any two states $s, s'$ in the set, there exists a DS policy such that $s'$ can be reached from $s$ in finite time with non-zero probability; and another set of states all of which are transient under any DS policy. A stronger condition is *unichain*, where for any DS policy, the induced Markov process is also unichain (i.e., has one recurrent class).

Under the usual definition of transition rates, we have $q(s,a,s) = -\sum_{s'\in\mathcal{S}:s'\neq s} q(s,a,s')$; hence, we do not generally consider a state to be able to transition to itself. This is,

even if one could consider some event with an exponentially distributed time that would technically cause state $s$ to transition to itself, this stochastic information is lost in the $q$ function, as the positive "outbound" and negative "inbound" transition rates of a self-transition would cancel out. However, we argue that it can be useful to keep this information explicitly encoded. To this end we define the *CTMDP with self-transitions* as a 4-tuple:

$$\mathfrak{C} = \left\langle \mathcal{S}, (\mathcal{A}(s))_{s\in\mathcal{S}}, q^+ : \mathcal{SA} \times \mathcal{S} \to \mathbb{R}_{\geq 0}, \boldsymbol{c} : \mathcal{SA} \to \mathbb{R}^k \right\rangle,$$

where the state space, action space, and cost functions are all defined as before, but we now define the *generalised transition rates* $q^+(s, a, s')$, satisfying $0 \leq q^+(s, a, s) < +\infty$, as the positive transition rate from state $s$ to state $s'$ when taking action $a$. This allows the explicit representation of transitions rates from a state back to itself. From the generalised transition rates, we can define the *negative generalised transition rates* as $q^-(s, a, s') := -\sum_{s''\in\mathcal{S}} q^+(s, a, s'')\mathbb{I}\{s = s'\}$ as the negative outbound flows of the transition rates. We can then recover the standard transition rates $q(s, a, s') := q^+(s, a, s') + q^-(s, a, s')$, where the rates in standard form satisfy:

$$\begin{aligned} +\infty > q(s, a, s') \geq 0, \text{ for all } s' \neq s, \\ -\infty < q(s, a, s') \leq 0, \text{ for } s' = s, \\ \sum_{s'\in\mathcal{S}} q(s, a, s') = 0. \end{aligned}$$

As such, all transitions from a state back to itself are ignored by any analysis that relies only on the $q$ function, and a standard CTMDP is recovered. It is also possible to go in the other direction by starting with a standard CTMDP with transition rates $q(s, a, s')$ and choosing $q^+(s, a, s')$ such that $q^+(s, a, s') = q(s, a, s')$ for any $s, a, s'$ such that $s \neq s'$, and $q^+(s, a, s)$ can then take on any positive value.

Upon first inspection, the notion of generalised transition rates may seem superflu-

ous. However, we shall use it to define a range of useful equivalence classes over states and state-action pairs. We give the following remark to demonstrate what is meant by saying that two states are equivalent.

**Remark 5.2.1.** *Consider the following two-state CTMC in standard form:*

$$Q = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}.$$

*We argue that this transition rate matrix can be obtained from the following positive transition rate matrix and its subsequent negative transition rate matrix:*

$$Q^+ = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, Q^- = \begin{bmatrix} -2 & 0 \\ 0 & -2 \end{bmatrix}.$$

*Here, it is easy to see that the two states are in some way "equivalent", in that they transition back to themselves and to each other at the same rate, but this information gets lost in the standard form. This notion of equivalence will be rigorously developed later.*

For any unichain CTMDP, we consider the infinite-horizon long-run average multi-objective LP formulation following Guo and Hernández-Lerma (2009), which we call

the Original Problem (OP):

$$\text{(OP)} \qquad \min_{\boldsymbol{\pi}} g_1 = \sum_{(s,a)\in\mathcal{SA}} c_1(s,a)\pi(s,a) \tag{5.2.1}$$

$$\vdots$$

$$\min_{\pi} g_K = \sum_{(s,a)\in\mathcal{SA}} c_K(s,a)\pi(s,a) \tag{5.2.2}$$

$$\text{s.t.} \sum_{(s,a)\in\mathcal{SA}} q(s,a,s')\pi(s,a) = 0, \text{ for all } s' \in \mathcal{S}, \tag{5.2.3}$$

$$\sum_{(s,a)\in\mathcal{SA}} \pi(s,a) = 1, \tag{5.2.4}$$

$$\pi(s,a) \geq 0, \text{ for all } (s,a) \in \mathcal{SA}. \tag{5.2.5}$$

Under a mixed-objectives scalarisation, this formulation is also valid for weakly-communicating CTMDPs, as such CTMDPs always permit a unichain optimal policy (Puterman, 2014, Chapter 9). Lines (5.2.1) to (5.2.2) give the $K$ objectives of the problem. Constraints (5.2.3) give the CTMDP balance equations which enforce the stochastic dynamics of the problem, and constraints (5.2.4)-(5.2.5) ensure that $\pi$ is a valid probability distribution. For $K > 1$, this is a multi-objective problem for which we need to consider the Pareto front. A Pareto-optimal solution may, in general, be a *randomized policy*, where for any recurrent state $s$ the probability of taking action $a$ in state $s$ is given by $\mu(s,a) = \pi(s,a)/\sum_{a'\in\mathcal{A}(s)} \pi(s,a')$ (Roijers et al., 2013). These policies correspond to non-basic feasible solutions of OP. Despite this, there always exists a subset of the Pareto-optimal policies which are deterministic (i.e., basic), and our analysis focuses on such policies. This is because these policies are likely to be more intuitive, easier to implement, and more explainable to the decision-maker than randomized policies. As such, we argue that the goal of finding all DS policies that are Pareto-optimal with respect to the set of stochastic stationary policies is a reasonable one, as such a set still offers great flexibility to the decision maker. The rest of this subsection is dedicated

to the development of a theory of *equivalence* over the states and state-action pairs of CTMDPs with self-transitions. This will be useful later for the analysis of impulsive CTMDPs, which can have many equivalent states or state-action pairs.

### Theory

We first define what it means for states or state-action pairs to be equivalent. To be clear, we are *not* defining the well-known equivalence relations that give the communicating classes commonly studied for Markov chains, but a different notion of equivalence that is needed for some of the analysis in this chapter.

**Definition 5.2.1** (Equivalence)**.** *For any finite-state finite-action CTMDP with self-transitions, with vector costs* $\boldsymbol{c}(s,a)$ *and generalised transition rates* $q^+$*, we say that two state-action pairs* $(s,a)$ *and* $(s',a')$ *are:*

*(i)* stochastically equivalent *if* $q^+(s,a,s'') = q^+(s',a',s'')$ *for all states* $s''$*, and write* $(s,a) \sim^{q^+} (s',a')$*,*

*(ii)* cost equivalent *if* $\boldsymbol{c}(s,a) = \boldsymbol{c}(s',a')$*, and write* $(s,a) \sim^{\boldsymbol{c}} (s',a')$*,*

*(iii)* totally equivalent *if* $(s,a) \sim^{q^+} (s',a')$ *and* $(s,a) \sim^{c} (s',a')$*, and write* $(s,a) \sim (s',a')$*.*

*Additionally, for any DS policy* $\mu$*, we say that two states* $s, s'$ *are:*

*(i)* stochastically equivalent *if* $(s,\mu(s)) \sim^{q^+} (s',\mu(s'))$*, and write* $s \sim_{\mu}^{q^+} s'$*,*

*(ii)* cost equivalent *if* $(s,\mu(s)) \sim^{\boldsymbol{c}} (s',\mu(s'))$*, and write* $s \sim_{\mu}^{\boldsymbol{c}} s'$*,*

*(iii)* totally equivalent *if* $(s,\mu(s)) \sim (s',\mu(s'))$*, and write* $s \sim_{\mu} s'$*.*

*Finally, for any equivalence relation* $R$ *over the feasible state-action space* $\mathcal{SA}$ *and any DS policy* $\mu$*, we can define the* $\mu$-adapted equivalence relation $R_\mu$ *over the state space* $\mathcal{S}$ *as* $s \, R_\mu \, s' \iff (s,\mu(s)) \, R \, (s',\mu(s'))$

As these are all based on equalities, it is immediately evident that the binary relations $\sim^{q^+}, \sim^{\mathcal{C}}, \sim$, and their $\mu$-adapted counterparts are all equivalence relations. We can therefore define the equivalence classes $\widehat{s,a}$ under any of these relations and the partitions $\widehat{\mathcal{SA}}$ of $\mathcal{SA}$ induced by those equivalence relations. For any policy $\mu$, we can also define the equivalence classes $\widehat{s}$ under any of the $\mu$-adapted relations and the partitions $\widehat{\mathcal{S}}$ of $\mathcal{S}$ induced by those equivalence relations.

Now, under any DS policy $\mu$, we wish to construct and analyse the stochastic process formed by mapping the state of the chain to its stochastic equivalence class. In fact, more generally than this, we can construct and analyse the CTMC formed from any equivalence relation $R$ that is *finer than* stochastic equivalence, i.e., $s\,R\,s' \implies s \sim_\mu^{q^+} s'$. Following Kemeny and Snell (1976), we can say that a controlled (discrete-time) Markov chain is *strongly lumpable* with respect to the partition of $\mathcal{S}$ induced by $R$, where a Markov chain over state space $\mathcal{S}$ is said to be strongly lumpable with respect to a partition $\widehat{\mathcal{S}}$ of $\mathcal{S}$ if:

$$\sum_{s\in\widehat{s}} p(s',s) = \sum_{s\in\widehat{s}} p(s'',s), \text{ for all } s',s''\in\widehat{s}', \text{ and for all } \widehat{s},\widehat{s}'\in\widehat{\mathcal{S}}.$$

This definition extends naturally to the continuous-time case via uniformisation. As such, the following Proposition 5.2.1, Corollary 5.2.1, and Proposition 5.2.2 all immediately follow; however, we provide direct alternative proofs in the appendices. We define $q_\mu^+(s,s') := q^+(s,\mu(s),s')$ and recognise that this value depends only on the class of $s$ with respect to $R$, and therefore we can write $q_\mu^+(\widehat{s},s')$ without loss of generality. We then define $q_\mu^+(\widehat{s},\widehat{s}') := \sum_{s''\in\widehat{s}'} q_\mu^+(\widehat{s},s'')$, which intuitively means that we expect the generalised transition rate from class $\widehat{s}$ to $\widehat{s}'$ to be the cumulative generalised transition

rate across all members of $\widehat{s}$ to any member of $\widehat{s}'$. We therefore also define:

$$q_\mu^-(\widehat{s}) := -\sum_{\widehat{s}' \in \widehat{\mathcal{S}}} q_\mu^+(\widehat{s}, \widehat{s}')$$
$$q_\mu^-(\widehat{s}, \widehat{s}') := \mathbb{I}\{\widehat{s} = \widehat{s}'\} q_\mu^-(\widehat{s}),$$

to give the negative generalised transition rates for any $\widehat{s}, \widehat{s}' \in \widehat{\mathcal{S}}$. We of course also define $q_\mu(\widehat{s}, \widehat{s}') := q_\mu^+(\widehat{s}, \widehat{s}') + q_\mu^-(\widehat{s}, \widehat{s}')$ to get the transition rates in standard form. Now, let $S^\mu(t)$ be the CTMC induced by a CTMDP with self-transitions controlled by a DS policy $\mu$, and let $\{S_k^\mu\}_{k=1}^\infty$ be $S^\mu(t)$ uniformised using some uniformisation parameter $\Delta$, following Serfozo (1979). We want to investigate the behaviour of the equivalence classes of this process with respect to any equivalence relation $R$ over $\mathcal{S}$ which is as strong as $\sim_\mu^{q^+}$, i.e., $sRs' \implies s \sim_\mu^{q^+} s'$. In other words, we wish to investigate the process that has been lumped with respect to the partition induced by such $R$. Define $\widehat{S_k^\mu} := \varphi_R(S_k^\mu)$, where $\varphi_R$ is the *canonical surjection* of $R$ (i.e., the function that maps an element to its equivalence class under $R$), so that $\widehat{S_k^\mu}$ is the stochastic process of the equivalence classes of the controlled CTMDP.

**Proposition 5.2.1.** $\left\{\widehat{S_k^\mu}\right\}_{k=1}^\infty$ *is a Markov chain over* $\widehat{\mathcal{S}}$ *with transition probabilities:*

$$p_\Delta(\widehat{s}, \widehat{s}') = \begin{cases} 1 + q(\widehat{s}, \widehat{s})\Delta, & \widehat{s}' = \widehat{s}, \\ q(\widehat{s}, \widehat{s}')\Delta, & \textit{otherwise.} \end{cases}$$

*Proof.* Follows immediately from lumpability with respect to $R$. See Section B.2.1 for a direct proof. □

We can also prove a similar equivalence property in continuous time. Under an equivalence relation $R$ as previously specified, let $\widehat{S^\mu} := \varphi_R \circ S^\mu$, be the stochastic process of the equivalence classes of $S^\mu$, where we recall $S^\mu$ is the CTMC induced by a CTMDP with self-transitions controlled by a DS policy $\mu$.

**Corollary 5.2.1.** *The stochastic process $\widehat{S^\mu}(t)$ is a continuous-time Markov chain over $\widehat{\mathcal{S}}$, equivalent to $\{\widehat{S_k^\mu}\}_{k=1}^\infty$ under uniformisation, with transition rates given by $q(\widehat{s},\widehat{s}')$ as previously defined.*

*Proof.* Follows immediately from lumpability with respect to $R$. See Section B.2.2 for a direct proof. □

We refer to the CTMC $\widehat{S^\mu}$ as the *class-aggregated* CTMC. We now wish to find the relationship between the any stationary distribution of the original controlled CTMDP, $\boldsymbol{\pi}^\mu$, and a stationary distribution of our class-aggregated CTMC.

**Proposition 5.2.2.** *If $\boldsymbol{\pi}^\mu$ is a stationary distribution for a CTMC, then a class-aggregated CTMC $\widehat{S}^\mu$ under a suitable equivalence relation $R$ has stationary distribution $\widehat{\boldsymbol{\pi}^\mu}$ satisfying $\widehat{\pi^\mu}(\widehat{s}) = \sum_{s\in\widehat{s}} \pi^\mu(s)$.*

*Proof.* Follows immediately from lumpability with respect to $R$. See Section B.2.3 for a direct proof. □

This proposition can naturally be extended from the setting of controlled CTMDPs to the solutions $\pi(s,a)$ of OP that correspond to DS policies. Let $\mu$ be such a policy and let $R$ be an equivalence relation as strong as $\sim^{q^+}$ with equivalence classes $\widehat{s,a}$, and let $R_\mu$ be its $\mu$-adapted equivalence relation with classes $\widehat{s}$. We define the *$R$-aggregated state-action frequencies* as $\widehat{\pi}(\widehat{s,a}) := \sum_{(s,a)\in\widehat{s,a}} \pi(s,a)$.

**Lemma 5.2.1.** *Let $\pi(s,a)$ be the basic feasible solution to the original problem* OP *corresponding to $\mu$, so $\pi(s,\mu(s)) = \pi^\mu(s)$ and $\pi(s,a) = 0$ for $a \neq \mu(s)$. The $R$-aggregated state-action frequencies satisfy the following equality:*

$$\widehat{\pi}(\widehat{s,a}) = \begin{cases} \widehat{\pi^\mu}(\widehat{s}), & \text{if there exists } (s',a') \in \widehat{s,a} \text{ such that } a' = \mu(s'), \\ 0, & \text{otherwise.} \end{cases}$$

*Proof.* See Section B.2.4. □

In the same way that generalised transition rates only depend on stochastic equivalence classes, cost rates $\boldsymbol{c}(s)$ depend only on the cost equivalence classes of $s$, so we can write $\boldsymbol{c}(\widehat{s,a}) := \boldsymbol{c}(s,a)$ for any $(s,a) \in \widehat{s,a}$, and for a fixed policy $\mu$ we can write $\boldsymbol{c}^\mu(\widehat{s}) := \boldsymbol{c}^\mu(s)$ for any $s \in \widehat{s}$. We can therefore associate these costs with the Markov chain $\widehat{S^\mu}$ and obtain a corollary to Proposition 5.2.2.

**Corollary 5.2.2.** *Consider a CTMDP with self-transitions $\mathfrak{C}$. Let $\widehat{\mathcal{S}}$ be the partition of the state space $\mathcal{S}$ with respect to an equivalence relation $R$ that is as strong as $\sim_\mu$, i.e., $\mu$-adapted total equivalence. The class-aggregated CTMC has the same long-run average cost as the original controlled CTMDP.*

*Proof.* We see that:

$$\sum_{s\in\mathcal{S}} \boldsymbol{c}^\mu(s)\pi^\mu(s) = \sum_{\widehat{s}\in\widehat{S}} \sum_{s\in\widehat{s}} \boldsymbol{c}^\mu(\widehat{s})\pi^\mu(s) = \sum_{\widehat{s}\in\widehat{S}} \boldsymbol{c}(\widehat{s})\widehat{\pi^\mu}(\widehat{s}),$$

so both the controlled CTMDP and the new CTMC with costs have the same long-run average costs. □

Intuitively, we have found that for any fixed DS policy, we can aggregate together any states with the same stochastic dynamics and costs, and the resulting aggregated CTMC behaves as desired. Section 5.2.2 now follows with a formulation for CTMDPs with impulsive control.

### 5.2.2 Impulsive CTMDPs

#### Model

We wish to model *impulsive control*, meaning that actions induce instantaneous transitions, and many such actions can be taken back-to-back in the same instant. This requires the development of some additional structure and assumptions on the CTMDP model:

1. We assume that all states $s$ have some "zero" action $0 \in \mathcal{A}(s)$, and this is the only action that allows time to advance, i.e., it is the one gradual action for that state, using the terminology of Dufour and Piunovskiy (2015). All other actions will be impulsive.

2. We require all state-action pairs $(s, a)$ with non-zero $a$ to "emulate" a unique "post-decision" state denoted by $s \oplus a \in \mathcal{S}$ under the zero action for that state, where $\oplus : \mathcal{SA} \to \mathcal{S}$ is a binary operation satisfying $s \oplus 0 = s$ for all $s$. By emulate, we mean the following relations are true:

$$q^+(s, a, s') = q^+(s \oplus a, 0, s'), \tag{5.2.6}$$

$$\boldsymbol{c}(s, a) = \boldsymbol{c}(s \oplus a, 0). \tag{5.2.7}$$

3. For any state $s$ we have $a_1 \in \mathcal{A}(s)$ and some $a_2 \in \mathcal{A}(s \oplus a_1)$, then there exists a unique action which we denote $a_1 \cdot a_2$, and this action must satisfy $s \oplus (a_1 \cdot a_2) = (s \oplus a_1) \oplus a_2$.

Assumption (1) ensures that there is only one way to allow "time to pass" in any given state, i.e., to stay in that state, as we are assuming that all possible controls or interventions are strictly impulsive. Generalisation to multiple gradual actions may be possible, but this is beyond the scope of this work. Assumption (2) emulates one-step impulsive control by ensuring that any state-action pair has the same stochastic dynamics and costs as some "target" state under that state's gradual action. As such, the process does not *truly* instantaneously transition upon taking an impulsive action, but otherwise acts like it has. This notion of a post-decision state can be thought of as the continuous-time equivalent of the post-decision states found in Powell (2011), which for some problems allows us to consider the "state" of the system after an action has been taken, but before the next random transition has been observed. Additionally, as a result of the relations over $q$ and $\boldsymbol{c}$, it is enough to specify transition rates under the

zero action $q^+(s, s') := q^+(s, 0, s')$ and cost rates under the zero action $\boldsymbol{c}(s) := \boldsymbol{c}(s, 0)$. Assumption (3) then ensures that the instantaneous sequencing of multiple impulsive actions can always be emulated by a "sequenced" action. Again, the process does not truly jump between multiple states during that instant, but it otherwise acts as though it has landed in that final state. Without confusion, we can simplify the notion of Assumption (3) by writing $s \oplus a_1 \oplus a_2$ instead of $(s \oplus a_1) \oplus a_2$ or $s \oplus (a_1 \cdot a_2)$ whenever it is clear that $s$ is a state and $a_1$ and $a_2$ are actions. Additionally, we may also write $a_1 \oplus a_2$ instead of $a_1 \cdot a_2$. We additionally assume *associativity* of $\oplus$ over actions, that is $(a_1 \oplus a_2) \oplus a_3 = a_1 \oplus (a_2 \oplus a_3)$ always holds, so we can write $a_1 \oplus a_2 \oplus a_3$.

**Remark 5.2.2.** *If $\mathcal{A}(s) = \mathcal{A}$ for all $s \in \mathcal{S}$, we can say that $(\mathcal{A}, \oplus)$ is a* monoid *(i.e., group without inverses) that (right-)acts on $\mathcal{S}$ such that $q^+$ and $\boldsymbol{c}$ obey the monoid action in the expected way.*

Putting this all together, an impulsive CTMDP is a tuple:

$$\mathfrak{C} = \langle \mathcal{S}, (\mathcal{A}(s))_{s\in\mathcal{S}}, q^+ : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}, \boldsymbol{c} : \mathcal{S} \to \mathbb{R}^k, \oplus : \mathcal{S}\mathcal{A} \to \mathcal{S} \rangle,$$

from which a CTMDP with self-transitions can be recovered by ensuring $q^+(s, a, s')$ and $\boldsymbol{c}(s, a)$ follow the relations (5.2.6) and (5.2.7), respectively.

We now consider what an optimal policy for such a process might look like. If under an optimal policy $\mu$ we take action $a \neq 0$ under state $s$, then it seems intuitive that action 0 should be taken in state $s \oplus a$, regardless of whether this state is reached via an impulsive action or a gradual action. On the other hand, if an optimal policy takes an impulsive action $a'$ in state $s \oplus a$, then it makes sense that it should take action $a \oplus a'$ in state $s$. In other words, we expect optimal policies to choose actions that result in direct transitions to the most desirable states, rather than choosing actions leading to states that they wish to exit from immediately. We formalise this notion by defining the property of *coherence*.

**Definition 5.2.2** (Coherent Policy)**.** *Let $\mu$ be a DS policy for an impulsive CTMDP. We say $\mu$ is a* coherent policy *if, for every state $s$ such that $\mu(s) = a \neq 0$, we have $\mu(s \oplus a) = 0$.*

We will later prove that there always exists a coherent optimal policy, and that this policy can easily be constructed from a non-coherent DS optimal policy (if one exists). This will be useful because a generic solution algorithm for CTMDPs (linear programming, value iteration, etc.) may not always return a coherent policy, but to prove this we need to understand the behaviour of non-coherent policies. In general, it is possible for non-coherent policies not just to have $a_2 = \mu(s \oplus a_1) \neq 0$, but also to have $a_3 = \mu(s \oplus a_1 \oplus a_2) \neq 0$ and so on, forming a "chain" of non-zero impulsive actions. The next definition formalises this idea.

**Definition 5.2.3** (Chain)**.** *Let $\mu$ be a DS policy for an impulsive CTMDP. Let $\{s_n\}_{n=1}^{\infty}$ be the sequence of states defined by $s_1 = s$ and $s_n = s_{n-1} \oplus \mu(s_{n-1})$ for $n \geq 2$. Let $\{a_n\}_{n=1}^{\infty}$ be the sequence of actions defined by $a_n = \mu(s_n)$. We call $\{a_n\}_{n=1}^{\infty}$ the* chain of $s$ under $\mu$*, and denote it by $Chain^{\mu}(s)$. We call $\{s_n\}_{n=1}^{\infty}$ the* state chain of $s$ under $\mu$*. We also define the* degree *of a chain by the number of non-zero actions in the chain.*

Using the notion of a chain, we can equivalently define coherent policies as DS policies in which the degree of the chain for every state is at most one. A chain can take on only two possible forms: a finite number of non-zero actions followed by an infinite sequence of zeros (i.e., $\{a_1, \ldots, a_N, 0, 0, 0, \ldots\}$); or an infinite sequence of non-zero actions. This is due to the fact that if $\mu(s_n) = 0$ for some $n \geq 1$, then $s_{n+1} = s_n$ and hence $\mu(s_m) = \mu(s_n)$ for all $m \geq n$. The infinite sequence will always eventually enter into a loop, as the state space is finite and the sequence is infinite and deterministic, so it must eventually loop back on itself. If the state chain of $s$ repeats $s$ for index $n > 1$, we call $s$ a *looping state*. From this understanding of chains, we may suggest a method for constructing a coherent optimal policy from a non-coherent optimal policy, which we state as a proposition.

**Theorem 5.2.1.** *Let $\mu$ be a non-coherent optimal policy for an impulsive CTMDP, implying that there exists at least one state $s$ such that $\mu(s) \neq 0$ and $\mu(s \oplus \mu(s)) \neq 0$. A coherent optimal policy can be constructed following this two-step process:*

1. *Set* $\mu'(s) = \begin{cases} \mu(s), & s \text{ is not looping,} \\ 0, & s \text{ is looping.} \end{cases}$.
   *The policy $\mu'$ is optimal.*

2. *Set $\mu''(s) = a_1 \oplus \cdots \oplus a_{N(s)}$, where $\{a_n\}_{n=1}^{\infty}$ is the chain of $s$ under $\mu'$ and $N(s)$ is the degree of that chain. This policy $\mu''$ is optimal and coherent.*

In short, we claim that coherent optimal policies can be constructed from non-coherent optimal policies by eliminating loops and aggregating chains of actions together. We defer the proof of Theorem 5.2.1 to Section 5.2.2. However, we *do not* suggest that this extends well to sub-optimal policies that result from the use of approximate methods such as reinforcement learning or bespoke heuristics.

**Remark 5.2.3.** *Let $a_1$ be a sub-optimal action for $s$ and let $a_2$ be a sub-optimal action for $s \oplus a_1$. The bound on sub-optimality is multiplicatively worse for the sequenced action $a_1 \oplus a_2$. In general, the bound on the error grows exponentially in the degree of the sequencing.*

A more rigorous statement and proof of Remark 5.2.3 is given in Section B.2.5. As such, one should proceed with caution when considering the use of sequencing on sub-optimal non-coherent policies obtained from approximate methods (reinforcement learning, heuristics, etc.). The remainder of this subsection provides the necessary mathematical treatment of Theorem 5.2.1, alongside other analysis of impulsive CTMDPs.

### Theory

The following lemma concerns the relative action value functions $Q^{\mu}$.

**Lemma 5.2.2.** *Consider an impulsive CTMDP. For any policy $\mu$ with constant gain $g^\mu$ and any state-action pair $(s, a)$, we have $Q^\mu(s, a) = Q^\mu(s \oplus a, 0)$.*

*Proof.*

$$\begin{aligned} Q^\mu(s,a) &= (c(s,a) - g^\mu)t^+(s,a) + \mathbb{E}_{s' \sim q^+(s,a,\cdot)} h^\mu(s'), \text{ (by definition of } Q^\mu) \\ &= (c(s \oplus a, 0) - g^\mu)t^+(s \oplus a, 0) + \mathbb{E}_{s' \sim q^+(s \oplus a, 0, \cdot)} h^\mu(s'), \text{ (by impulsivity)} \\ &= Q^\mu(s \oplus a, 0), \end{aligned}$$

where $t^+(s,a) = 1/q^-(s,a,s)$ is the sojourn time of state $s$ under action $a$ when allowing for "dummy" transitions back to $s$, and $h^\mu$ is the bias (i.e., relative value function) of policy $\mu$. □

We wish to prove that, in the single objective case, there always exists a coherent policy that is optimal, and that this can be constructed by following Theorem 5.2.1. This then extends naturally to the multi-objective case, because for any *supported* Pareto-optimal DS policy $\mu$, there exists an objective weighting $\boldsymbol{w}$ such that $\mu$ is optimal for single-objective cost function $c_{\boldsymbol{w}}(s) = \boldsymbol{w} \cdot \boldsymbol{c}(s)$, so there will then exist a coherent policy $\mu'$ that is also optimal for this weighting, implying that $\mu'$ is Pareto-optimal. The approach will require the use of *sequencing operators*, which we define next.

**Definition 5.2.4** (Sequencing Operators)**.** *Let $\mathcal{M}$ be the space of DS policies for an impulsive CTMDP. We call a function $Seq_k : \mathcal{M} \to \mathcal{M}$ the* $k$th order sequencing operator *where $Seq_k\{\mu\}(s) = a_1 \oplus \cdots \oplus a_k$, and $\{a_n\}_{n=1}^\infty$ is the chain of $s$ under $\mu$. We call $Seq_k\{\mu\}$ the $k$th order sequenced policy.*

An equivalent and recursive definition is $\text{Seq}_1\{\mu\}(s) = \mu(s) \oplus \mu(s \oplus \mu(s))$, and $\text{Seq}_k\{\mu\}(s) = \text{Seq}_{k-1}\{\mu\}(s) \oplus \mu(s \oplus \text{Seq}_{k-1}\{\mu\}(s))$ for $k \geq 2$. This method of sequencing can be used to break loops and reduce chains to length 0 or 1, while maintaining optimality. To prove this, we must first prove that one step of sequencing maintains optimality.

**Lemma 5.2.3.** *Let $Q^*$ be the optimal $Q$ value function for a single-objective impulsive CTMDP (the choice of additive constant does not matter). If $a_1 \neq 0$ is optimal for some state $s$ and $a_2 \neq 0$ is optimal for $s \oplus a_1$, then $a_1 \oplus a_2$ is also optimal for $s$.*

*Proof.* Suppose that there exists a state $s$ such that $s, a_1, a_2$ are as described. By optimality, $Q^*(s, a_1) \leq Q^*(s, a)$ for all states $s$ and actions $a$. For any such $s$, we have $Q^*(s \oplus a_1, 0) = Q^*(s, a_1) \leq Q^*(s, a_1 \oplus a_2) = Q^*(s \oplus a_1 \oplus a_2, 0)$. However, because $a_2$ is optimal for $s \oplus a_1$, we also have $Q^*(s \oplus a_1 \oplus a_2, 0) = Q^*(s \oplus a_1, a_2) \leq Q^*(s \oplus a_1, 0)$. Therefore $Q^*(s \oplus a_1, 0) = Q^*(s \oplus a_1 \oplus a_2, 0)$ and hence $Q^*(s, a_1) = Q^*(s, a_1 \oplus a_2)$, so action $a_1 \oplus a_2$ is also optimal for $s$. □

We can use Lemma 5.2.3 to show that for any optimal policy $\mu$, its sequenced forms are also optimal.

**Proposition 5.2.3.** *Let $\mu$ be an optimal policy. Then $Seq_k\{\mu\}$ is an optimal policy for all $k$.*

*Proof.* We proceed by induction. It is immediate from Lemma 5.2.3 that $\text{Seq}_1\{\mu\}$ is optimal. Now assume $\text{Seq}_k\{\mu\}$ is optimal for some $k$. It follows that for any state $s$, $\text{Seq}_k\{\mu\}(s)$ is optimal for $s$ and $\mu(s \oplus \text{Seq}_k\{\mu\}(s))$ is optimal for $s \oplus \text{Seq}_k\{\mu\}(s)$. Hence, by Lemma 5.2.3, $\text{Seq}_k\{\mu\}(s) \oplus \mu(s \oplus \text{Seq}_k\{\mu\}(s))$ is also optimal for $s$, so by definition $\text{Seq}_{k+1}\{\mu\}(s)$ is optimal for $s$. Therefore, $\text{Seq}_{k+1}\{\mu\}$ is also an optimal policy. □

We can use Proposition 5.2.3 to show how to break loops and aggregate the actions in chains to obtain a coherent optimal policy.

**Proposition 5.2.4.** *Let $\mu$ be an optimal policy for an impulsive CTMDP, and let $s$ be a looping state with respect to $\mu$. Then action 0 is optimal for $s$.*

*Proof.* If $s$ is a looping state, then it reappears at some index $n > 1$ in its state chain, so $s \oplus a_1 \oplus \cdots \oplus a_{n-1} = s$ and therefore $a_1 \oplus \cdots \oplus a_{n-1} = 0$. Action $a_1 \oplus \cdots \oplus a_{n-1}$ is also optimal for $s$ because $\text{Seq}_{n-2}\{\mu\}$ is an optimal policy. Therefore 0 is optimal for $s$. □

As such, for any optimal policy $\mu$, the policy $\mu'$ described in Theorem 5.2.1 is also optimal. The second part of the claim is simply a corollary of Proposition 5.2.3.

**Corollary 5.2.3.** *Let $\mu$ be an optimal policy with no looping states, so all chains have a finite number of non-zero actions. Then, for any state $s$, $a_1 \oplus \cdots \oplus a_{N(s)}$ is optimal for state $s$, where the $a_n$ form the chain of $s$ with degree $N(s)$.*

Proposition 5.2.4 and Corollary 5.2.3 together provide a proof of Theorem 5.2.1. This confirms that we can sequence together actions from a non-coherent optimal policy to obtain a coherent optimal policy. As previously mentioned, this naturally extends to supported Pareto-optimality in the multi-objective case.

So far, we have shown that transition rates, costs, value functions, and policies behave as expected when impulsive actions are implemented in the suggested manner. However, extra care is needed when examining the long-run probability of being in a particular state. Consider the CTMC induced by an impulsive CTMDP controlled by a coherent policy $\mu$, denoted by $S^\mu(t)$. Let $\boldsymbol{\pi}^\mu$ be the stationary distribution, and suppose there exists a recurrent state $s$ such that $\mu(s) = a \neq 0$. As $s$ is recurrent, we have $\pi^\mu(s) > 0$. However, in order to emulate impulsive actions, it would be better to have $\pi^\mu(s) = 0$, as the policy $\mu$ chooses to exit from state $s$ immediately. We can also think of this non-zero $\pi^\mu(s)$ value as detracting from the probability of being in state $s \oplus a$, the instantaneous post-decision state. To resolve this issue, we must build upon the equivalence relations introduced in the previous subsection and investigate the probabilities with respect to the equivalence relations. In this way, the probability of being in a state with an impulsive action can be reconciled with the probability of being in its target state in a way that makes sense. These equivalence relations are also useful in the analysis of decomposable impulsive CTMDPs in Section 5.2.2. We proceed to define a unique sense of equivalence over impulsive CTMDPs.

**Definition 5.2.5** (Impulsive Equivalence)**.** *For any impulsive CTMDP, we say two feasible state-action pairs $(s_1, a_1), (s_2, a_2) \in \mathcal{SA}$ are* impulsively equivalent *if $s_1 \oplus a_1 =$*

*$s_2 \oplus a_2$, and write $(s_1, a_1) \sim_{\oplus} (s_2, a_2)$. For an impulsive CTMDP controlled by a DS policy $\mu$, we say two states $s_1$ and $s_2$ are* impulsively equivalent *if $(s_1, \mu(s_1)) \sim_{\oplus} (s_2, \mu(s_2))$, and write $s_1 \sim_{\oplus\mu} s_2$.*

An immediate consequence of this definition is that two impulsively equivalent states are also equivalent with respect to $\sim_\mu$. Impulsive equivalence is therefore a strictly stronger condition than total equivalence. We demonstrate this using the following example.

**Example 5.2.1.** *Consider a controlled impulsive CTMDP with state space $\mathcal{S} = \{1, 2, 3, 4\}$ and action spaces:*

$$\mathcal{A}(1) = \mathcal{A}(2) = \{0, 1\},\ \mathcal{A}(3) = \mathcal{A}(4) = 0.$$

*Let $\mu(1) = \mu(2) = 1$, $1 \oplus 1 = 3$ and $2 \oplus 1 = 4$. Suppose all costs are 0, so all states are trivially cost equivalent. Suppose $q^+(3, 0, 3) = q^+(3, 0, 4) = q^+(4, 0, 3) = q^+(4, 0, 4) = 1$. Then all states are stochastically equivalent under $\mu$ and hence are all equivalent under $\mu$. However, state-action pairs $(1, 1)$ and $(2, 1)$ are not impulsively equivalent because $1 \oplus 1 \neq 2 \oplus 1$.*

We also have that $\boldsymbol{c}(s_1, a_1) = \boldsymbol{c}(s_2, a_2)$ for impulsively equivalent pairs $(s_1, a_1)$ and $(s_2, a_2)$, so we can write $\boldsymbol{c}(\widehat{s_1, a_1})$ without any ambiguity. We can immediately connect this sense of equivalence to the notion of coherence.

**Lemma 5.2.4.** *Let $\mu$ be a coherent policy. Then $s \sim_{\oplus\mu} s \oplus \mu(s)$.*

*Proof.* We see that $(s, \mu(s)) \sim_{\oplus} (s \oplus \mu(s), 0) = (s \oplus \mu(s), \mu(s \oplus \mu(s)))$, so $s \sim_{\oplus\mu} s \oplus (\mu(s))$. □

This intuitively tells us that any state is equivalent to its post-decision state under a coherent policy. This allows us to aggregate all states with the same post-decision state (including the post-decision state itself) into one equivalence class, and as such

we can report the values $\widehat{\pi^\mu}(\widehat{s})$ for this post-decision state as if the actions were truly instantaneous. Next, we show that this kind of post-decision state in a class is unique.

**Lemma 5.2.5.** *Let $\mu$ be a coherent policy, and let $\widehat{s}$ be an equivalence class with respect to $\sim_{\oplus\mu}$. There is exactly one $s \in \widehat{s}$ such that $\mu(s) = 0$, which we call the post-decision state of the class.*

*Proof.* We first prove existence of such a state. The class $\widehat{s}$ is a non-empty set and therefore contains some state $s'$. It follows that either $\mu(s') = 0$, so such a state exists, or $\mu(s') \neq 0$ and therefore the class also contains $s' \oplus \mu(s')$ by Lemma 5.2.4, which by definition of coherence satisfies $\mu(s' \oplus \mu(s')) = 0$. To show uniqueness, suppose there exist two states $s_1, s_2 \in \widehat{s}$ such that $\mu(s_1) = \mu(s_2) = 0$. Then since $s_1 \sim_{\oplus\mu} s_2$ we have $(s_1, \mu(s_1)) \sim_\oplus (s_2, \mu(s_2))$ and hence $(s_1, 0) \sim_\oplus (s_2, 0)$. Therefore $s_1 \oplus 0 = s_2 \oplus 0$, implying that $s_1 = s_2$. □

We can interpret the value $\widehat{\pi^\mu}(\widehat{s})$ as the probability of being in the one state in $\widehat{s}$ in which we take the gradual action, rather than viewing this probability as being 'distributed' between the states which emulate this state impulsively. To be more precise, by Corollary 5.2.1, we know that if $S^\mu$ is the CTMC induced by controlling an impulsive CTMDP, then the class-aggregated process $\widehat{S^\mu}$ is also a CTMC, and $\widehat{\pi^\mu}(\widehat{s})$ is the stationary distribution of that CTMC. We can then relabel the $\widehat{s}$ to their post-decision states to obtain a CTMC that always transitions between post-decision states, such that no time is ever spent in states where a non-zero action is taken. As such, the desired behaviour of instant transitions from impulsivity is satisfied up to an equivalence relation when the impulsive CTMDP is controlled by a coherent policy, as desired. This concludes our general results for impulsive CTMDPs, and Section 5.2.3 now follows with the introduction and analysis of decomposable impulsive CTMDPs.

### 5.2.3 Decomposable Impulsive CTMDPs

#### Model

We build towards the definition of a decomposable impulsive CTMDP in two stages. The first stage deals with the stochastic dynamics of the system whilst ignoring the costs, and the second stage deals with the structure of the cost function. In order to define a problem that can be broken into smaller problems, we take the approach of defining the smaller problems first and then combining these together. This motivates our first definition.

**Definition 5.2.6** (Cost-Free Direct Product of Impulsive CTMDPs)**.** *Let* $\{\mathfrak{C}_i\}_{i=1}^{I}$ *be a finite indexed set of impulsive CTMDPs* $\mathfrak{C}_i = \langle \mathcal{S}_i, (\mathcal{A}_i(s_i))_{s_i \in \mathcal{S}_i}, q_i^+ : \mathcal{S}_i \times \mathcal{S}_i \to \mathbb{R}_{\geq 0}, \oplus_i : \mathcal{S}\mathcal{A}_i \to \mathcal{S}_i \rangle$*. We define the* cost-free direct product *of these impulsive CTMDPs as the costless impulsive CTMDP:*

$$\bigotimes_{i=1}^{I} \mathfrak{C}_i := \langle \mathcal{S}, (\mathcal{A}(s))_{s \in \mathcal{S}}, q^+ : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}, \oplus : \mathcal{S}\mathcal{A} \to \mathcal{S} \rangle,$$

*satisfying:*

$$\begin{aligned}
\mathcal{S} &= \bigtimes_{i=1}^{I} \mathcal{S}_i \\
\mathcal{A}(\boldsymbol{s}) &= \bigtimes_{i=1}^{I} \mathcal{A}_i([\boldsymbol{s}]_i) \\
q^+(\boldsymbol{s}, \boldsymbol{s}') &= \begin{cases} q_i^+([\boldsymbol{s}]_i, s_i'), & \boldsymbol{s}' = ([\boldsymbol{s}]_1, \ldots, s_i', \ldots, [\boldsymbol{s}]_I) \text{ for some } i, \\ 0, & \text{otherwise,} \end{cases} \\
\boldsymbol{s} \oplus \boldsymbol{a} &= ([\boldsymbol{s}]_1 \oplus_1 [\boldsymbol{a}]_1, \ldots, [\boldsymbol{s}]_I \oplus_I [\boldsymbol{a}]_I).
\end{aligned}$$

*Additionally, we refer to* $\mathfrak{C}_i$ *as the* $i$th compartment *of the product CTMDP.*

By construction, this CTMDP is also impulsive, and the state space, action spaces

and impulsive operator take the expected forms. The function $q^+$ requires more explanation: the structure of $q^+$ ensures that only one of the compartments can update upon a transition, whilst the others remain the same. This is because the rate of such a transition should depend only on the state (or "sub-state") of that compartment, and not on the rest of the composite state. Now, using this direct product, we can provide a further definition.

**Definition 5.2.7** (Stochastic Decomposability)**.** *An impulsive CTMDP*

$$\mathfrak{C} = \langle \mathcal{S}, (\mathcal{A}(s))_{s\in\mathcal{S}}, q^+ : \mathcal{S}\times\mathcal{S} \to \mathbb{R}_{\geq 0}, \boldsymbol{c} : \mathcal{SA} \to \mathbb{R}^k, \oplus : \mathcal{SA} \to \mathcal{S} \rangle$$

*is called* stochastically decomposable with $I$ compartments *if the tuple* $\langle \mathcal{S}, (\mathcal{A}(s))_{s\in\mathcal{S}}, q^+, \oplus \rangle$ *can be expressed as a cost-free direct product of impulsive CTMDPs* $\{\mathfrak{C}_i\}_{i=1}^I$*. A choice of such sub-CTMDPs then gives the* compartments *of the stochastically decomposable impulsive CTMDP.*

Trivially, any impulsive CTMDP is stochastically decomposable with one compartment. Also, the choice of compartmentalization is not unique. For example, the cost-free direct product of three compartments could be decomposed back into just two compartments, with two of the three original compartments remaining joined together. Before we turn our attention to the structure of the cost functions for a decomposable impulsive CTMDP, we must first consider a certain type of policy to be used for controlling such a process. Heuristically, we might think that it would make sense to control the compartments of the CTMDP separately, only making decisions for a compartment based on the sub-state and not the whole state. We argue that taking this approach could allow us to optimise the compartment CTMDPs separately, resulting in subproblems that are far easier to solve. We formalise this idea by introducing *separable policies.*

**Definition 5.2.8** (Separable Policy)**.** *Let $\mu$ be a DS policy for a stochastically decom-*

*posable impulsive CTMDP with compartments* $\{\mathfrak{C}_i\}_{i=1}^I$. *We say that* $\mu$ *is a* separable policy *if there exist functions* $\mu_i$ *such that, for all* $\boldsymbol{s} \in \mathcal{S}$, *we can write* $\mu(\boldsymbol{s}) = (\mu_1([\boldsymbol{s}]_1), \mu_2([\boldsymbol{s}]_2), \ldots, \mu_I([\boldsymbol{s}]_I))$. *That is, the action taken on compartment* $i$ *depends only on the state of compartment* $i$.

**Remark 5.2.4.** *The flexibility of separable policies relies on the* choice *of decomposition. As such, we advise caution when considering the choice of decomposition when limiting to such policies. It is not always necessarily best to decompose a problem as much as possible, as heuristics based on the use of separable policies may sometimes perform better when certain compartments are left non-decomposed.*

Next, we discuss decomposability of the cost function. We do not wish to restrict attention to cost functions that are simply linear combinations of cost functions from the compartment CTMDPs. The following definition allows for a broader range of cost functions.

**Definition 5.2.9** (Decomposable Cost Function)**.** *Let* $\mathfrak{C}$ *be a stochastically decomposable impulsive CTMDP with* $I$ *compartments and* $K$ *objectives. Let* $\boldsymbol{g} : [0,1]^{|\mathcal{SA}|} \to \mathbb{R}^K, \boldsymbol{g}(\pi) = \sum_{s\in\mathcal{S}, a\in\mathcal{A}(s)} \boldsymbol{c}(s \oplus a)\pi(s,a)$ *be the function mapping every solution of OP to its long-run average cost. The cost function* $c_k : \mathcal{S} \to \mathbb{R}$ *is a* decomposable cost function *if there exists an increasing function* $f_k : \mathbb{R}^I \to \mathbb{R}$ *and functions* $c_{i,k} : \mathcal{S}_i \to \mathbb{R}$ *for* $i \in \{1, \ldots, I\}$, *which we call the* $k$th sub-state costs, *such that for every* $\boldsymbol{\pi}$ *corresponding to a coherent separable policy* $\mu$, *we have:*

$$g_k(\pi) = f_k\left(\sum_{(s_1,a_1)\in\mathcal{SA}_1} c_{1,k}(s_1 \oplus a_1)\pi_1(s_1,a_1), \ldots, \sum_{(s_I,a_I)\in\mathcal{SA}_I} c_{I,k}(s_I \oplus a_I)\pi_I(s_I,a_I)\right),$$

*where* $\pi_i(s_i,a_i) := \sum_{\boldsymbol{s},\boldsymbol{a}:[\boldsymbol{s},\boldsymbol{a}]_i=(s_i,a_i)} \pi(\boldsymbol{s},\boldsymbol{a})$ *is the probability that compartment* $i$ *is in sub-state* $s_i$ *and undergoing sub-action* $a_i$.

**Remark 5.2.5.** *We may write the right-hand-side of the above expression in shorthand*

*form as:*

$$f_k\left(\left(\sum_{(s_i,a_i)\in\mathcal{SA}_i} c_{i,k}(s_1 \oplus a_1)\pi_i(s_1,a_1)\right)_{i=1}^{I}\right).$$

The above definition is very general and quite verbose, so we discuss a few simple cases for the structure of a cost function and propose that each case implies decomposability.

**Definition 5.2.10** (Types of Cost Function)**.** *Let $\mathfrak{C}$ be a stochastically decomposable impulsive CTMDP with $I$ compartments and $K$ objectives. Then $c_k : \mathcal{S} \to \mathbb{R}$ is:*

- *an* additive cost function *if there exist functions $c_{i,k} : \mathcal{S}_i \to \mathbb{R}$ such that $c_k(\boldsymbol{s}) = \sum_{i=1}^{I} c_{i,k}([\boldsymbol{s}]_i)$.*
- *a* multiplicative cost function *if there exist functions $c_{i,k} : \mathcal{S}_i \to \mathbb{R}$ such that $c_k(\boldsymbol{s}) = \prod_{i=1}^{I} c_{i,k}([\boldsymbol{s}]_i)$.*
- *a* logical-OR cost function *if there exist subsets $E_i \subset \mathcal{S}_i$ for each $i$ such that $c_k(\boldsymbol{s}) = \mathbb{I}\left\{\bigvee_{i=1}^{I}[\boldsymbol{s}]_i \in E_i\right\}$.*

The additive cost function captures the simple idea of adding the costs of individual compartments, which often makes sense when the cost function represents monetary costs. However cost functions can sometimes have alternative interpretations. The logical-OR cost function is particularly useful if there is a set of states in each compartment that should be avoided, and therefore we aim to avoid any state of the whole system that contains such sub-states of compartments.

**Proposition 5.2.5.** *Additive cost functions, multiplicative cost functions, and logical-OR cost functions are all decomposable cost functions.*

*Proof.* See Section B.2.8 □

Note that these cost functions are not exhaustive, and there may be other useful types of decomposable cost functions. Next, we define the decomposable impulsive CTMDP.

**Definition 5.2.11** (Decomposable Impulsive CTMDP)**.** *An impulsive CTMDP* $\mathfrak{C}$ *is called a* decomposable impulsive CTMDP *if it is stochastically decomposable and each of its scalar cost functions* $c_k$ *is a decomposable cost function.*

This concludes the construction of the modelling framework. Those wishing to model a problem as a decomposable impulsive CTMDP may choose to construct their model directly and then prove that it is decomposable. Alternatively, they may choose to construct the compartment CTMDPs first and then use the direct product to construct the full model, and define cost functions over this model. The remainder of this subsection is dedicated to the analysis of stochastically decomposable impulsive CTMDPs.

### Theory

The following lemma links the concepts of separable policies and coherence.

**Lemma 5.2.6.** *A separable policy* $\mu$ *is coherent if and only if each subpolicy* $\mu_i$ *is coherent.*

*Proof.* Let $\mu$ be coherent and separable, so there exist policies $\mu_i : \mathcal{S}_i \to \mathcal{A}_i$ such that $\mu(\boldsymbol{s}) = (\mu_1([\boldsymbol{s}]_1), \ldots, \mu_I([\boldsymbol{s}]_I))$ for all $i$. We also have that $\mu(\boldsymbol{s} \oplus \mu(\boldsymbol{s})) = 0$ for all $s$. Now:

$$\begin{aligned}
\mu(\boldsymbol{s} \oplus \mu(\boldsymbol{s})) = \mathbf{0} \text{ for all } \boldsymbol{s} &\iff [\mu(\boldsymbol{s} \oplus \mu(\boldsymbol{s}))]_i = 0 \text{ for all } \boldsymbol{s}, i \\
&\iff \mu_i([\boldsymbol{s} \oplus \mu(\boldsymbol{s})]_i) = 0 \text{ for all } \boldsymbol{s}, i \\
&\iff \mu_i([\boldsymbol{s}]_i \oplus \mu_i([\boldsymbol{s}]_i)) = 0 \text{ for all } \boldsymbol{s}, i \\
&\iff \mu_i(v \oplus \mu_i(v)) = 0 \text{ for all } i, v \in \mathcal{S}_i.
\end{aligned}$$

So the bi-implication holds. □

Note that we now start to use $v$ for generic compartment states and $b$ for generic

compartment actions, and we abbreviate policies that are coherent and separable to *CS* policies. Now, for a choice of compartments $\{\mathfrak{C}_i\}_{i=1}^I$ for a stochastically decomposable impulsive CTMDP, we can form the impulsive equivalence relations over the state-action pairs of the compartment CTMDPs $\mathfrak{C}_i$, and we denote this relation by $\sim_{\oplus_i}$. This induces a partition over the compartment feasible state-action spaces $\mathcal{SA}_i$ which we denote $\widehat{\mathcal{SA}_i}$. Similarly, for any separable policy $\mu$, we can form impulsive equivalence relations over the states of the compartment CTMDPs using the sub-policies $\mu_i$, and denote these by $\sim_{\oplus\mu_i}$. In the following lemma and its corollary, we connect these two relations.

**Lemma 5.2.7.** *Let $(\boldsymbol{s}_1, \boldsymbol{a}_1), (\boldsymbol{s}_2, \boldsymbol{a}_2) \in \mathcal{SA}$ be two state-action pairs of a decomposable impulsive CTMDP. Then $([\boldsymbol{s}_1]_i, [\boldsymbol{a}_1]_i) \sim_{\oplus_i} ([\boldsymbol{s}_2]_i, [\boldsymbol{a}_2]_i)$ for all $i$ if and only if $(\boldsymbol{s}_1, \boldsymbol{a}_1) \sim_{\oplus} (\boldsymbol{s}_2, \boldsymbol{a}_2)$.*

*Proof.* We have $([\boldsymbol{s}_1]_i, [\boldsymbol{a}_1]_i) \sim_{\oplus_i} ([\boldsymbol{s}_2]_i, [\boldsymbol{a}_2]_i)$ for all $i$ if and only if $[\boldsymbol{s}_1]_i \oplus [\boldsymbol{a}_1]_i = [\boldsymbol{s}_2]_i \oplus [\boldsymbol{a}_2]_i$ for all $i$, which holds if and only if $\boldsymbol{s}_1 \oplus \boldsymbol{a}_1 = \boldsymbol{s}_2 \oplus \boldsymbol{a}_2$, which is identical to $(s_1, a_1) \sim_{\oplus} (s_2, a_2)$ by definition. □

**Corollary 5.2.4.** *For a separable policy $\mu$, let $\boldsymbol{s}_1, \boldsymbol{s}_2$ be two states of a decomposable impulsive CTMDP. Then $[\boldsymbol{s}_1]_i \sim_{\oplus\mu_i} [\boldsymbol{s}_2]_i$ for all $i$ if and only if $\boldsymbol{s}_1 \sim_{\oplus\mu} \boldsymbol{s}_2$.*

*Proof.* This is immediate from the definitions and Lemma 5.2.7. □

Next, we show how a controlled decomposable impulsive CTMDP can be decomposed by making use of the equivalence relations developed. We first prove this for the uniformised discrete-time case, and then extend to the continuous-time case.

**Proposition 5.2.6.** *Let $\{S_k^\mu\}_{k=1}^\infty$ be the discrete-time Markov chain obtained by uniformising (with parameter $\Delta$) the CTMC induced by controlling a stochastically decomposable impulsive CTMDP with a coherent separable policy $\mu$. Let $S_{i,k}^\mu = [S_k^\mu]_i$ be the random variable determined by the $i$th component, i.e., the state of compartment $i$, for*

*any $i$ at time $k$. Let $\widehat{S^{\mu}_{i,k}} := \varphi_i\left(S^{\mu}_{i,k}\right)$ for all $i$, where $\varphi_i$ is the canonical surjection of $\sim_{\oplus\mu_i}$. The processes $\left\{\widehat{S^{\mu}_{i,k}}\right\}_{k=1}^{\infty}$ for all $i$ are independent Markov chains.*

*Proof.* See Section B.2.7. □

We can also provide a continuous-time corollary of Proposition 5.2.6 that relates the continuous-time controlled compartment CTMDPs to the decomposed uniformised class-aggregated MDPs. Let $S^{\mu}(t)$ be the CTMC induced by controlling a stochastically decomposable impulsive CTMDP with a coherent separable policy $\mu$. Let $V^{\mu_i}(t)$ be the stochastic process determined by the $i$th compartment, i.e., the CTMDP $\mathfrak{C}_i$ controlled by $\mu_i$ (note that this is not the same as $[S^{\mu}(t)]_i$, as updates in the latter can be triggered by events in other compartments, whereas $V^{\mu_i}$ is a wholly independent process not influenced by exogenous information). Let $\widehat{V^{\mu_i}}(t)$ be the stochastic process defined by taking the $\sim_{\oplus\mu_i}$ equivalence classes of $V^{\mu_i}(t)$.

**Corollary 5.2.5.** *The process $\widehat{V^{\mu_i}}(t)$ is a continuous-time Markov chain that is equivalent in distribution to $\{S^{\mu}_{i,k}\}_{k=1}^{\infty}$ under uniformisation.*

*Proof.* This follows immediately from Corollary 5.2.1 and inspection of the transition probabilities derived in the proof for Proposition 5.2.6. □

The interpretation of the above corollary is somewhat subtle. First, we discuss the notion of $V^{\mu_i}(t)$ and $[S^{\mu}(t)]_i$ being distinct. Suppose that a two-compartment CTMDP $\mathfrak{C}$ being controlled by a coherent separable policy $\mu$ is in state $\boldsymbol{s} = (s_1, s_2)$, with a non-zero action being taken in both compartments. Upon the next transition, the process moves to a new state of the form $(s_1', s_2 \oplus a_2)$ or $(s_1 \oplus a_1, s_2')$, so the new state reflects an event in one of the compartments and reflects taking the impulsive action in the other. This is what the whole process $S^{\mu}(t)$ would tell us, meaning the state of one compartment can be updated to reflect the impulsive action being taken, when the event actually happened in the other compartment. As such, the process $[S^{\mu}(t)]_1$ is dependent on transition times in $[S^{\mu}(t)]_2$ and vice versa. However, this is not the case with $V^{\mu_i}(t)$,

the compartment CTMDP $\mathfrak{C}_i$ controlled by coherent policy $\mu_i$. Consider $V^{\mu_1}(t)$, the controlled first compartment, and suppose it is in state $s_1$ under action $a_1$. This process cannot be updated to $s_1 \oplus a_1$ by some exogenous event, and can only be updated by an actual event in its own compartment. However, when factoring in uniformisation and equivalence relations, $\widehat{V^{\mu_i}}(t)$ is equivalent to $\{S^{\mu}_{i,k}\}_{k=1}^{\infty}$ upon uniformisation. This further explains why coherent policies are needed. Without the condition of coherence, when $S^{\mu}(t)$ transitions from $(s_1, s_2)$ to $(s_1 \oplus a_1, s_2')$, one could technically take a new action for $s_1 \oplus a_1$, whereas this cannot be done for $V^{\mu_i}(t)$ because no transition is triggered. However, under a coherent policy, this issue never arises.

As these two processes have the same stochastic dynamics, we can without loss of generality consider the random variable $\widehat{S^{\mu}_i}$, which we define as the unique equilibrium distribution of both $\widehat{V^{\mu_i}}(t)$ and $\{\widehat{S^{\mu}_{i,k}}\}_{k=1}^{\infty}$ (note that while this may not exist nor be unique in general, it will exist and be unique given the initial distributions $\widehat{V^{\mu_i}}(0)$ or $\widehat{S^{\mu}_{i,0}}$, and the guarantee of aperiodicity due to continuous time). These are of course linked to the random variable $\widehat{S^{\mu}}$, the stationary distribution of the whole-state equivalence class process $\widehat{S^{\mu}}(t)$. The next corollary relates these two distributions.

**Corollary 5.2.6.** *The random variables $\widehat{S^{\mu}}$ and $\widehat{S^{\mu}_i}$ satisfy the following equalities:*

$$\mathbb{P}(\widehat{S^{\mu}} = \widehat{\boldsymbol{s}}) = \prod_{i=1}^{I} \mathbb{P}(\widehat{S^{\mu}_i} = [\widehat{\boldsymbol{s}}]_i), \text{ for all } \widehat{\boldsymbol{s}} \in \widehat{\mathcal{S}},$$

$$\mathbb{P}(\widehat{S^{\mu}_i} = \widehat{v}) = \sum_{\widehat{\boldsymbol{s}}:[\widehat{\boldsymbol{s}}]_i = \widehat{v}} \mathbb{P}(\widehat{S^{\mu}} = \widehat{\boldsymbol{s}}), \text{ for all } i \text{ and } \widehat{v} \in \widehat{\mathcal{S}}_i.$$

*Proof.* By Corollary 5.2.5, we know the $\widehat{S^{\mu}_i}$ are the stationary distributions of the discrete-time Markov chains $\widehat{S^{\mu}_{i,k}}$, and that the $i$ chains are independent. Firstly, we

see:

$$\begin{aligned}\mathbb{P}(\widehat{S^\mu} = \widehat{\boldsymbol{s}}) &= \mathbb{P}\left(\bigcap_{i=1}^{I}\left(\left[\widehat{S^\mu}\right]_i = [\widehat{\boldsymbol{s}}]_i\right)\right) \\ &= \prod_{i=1}^{I}\mathbb{P}(\widehat{S_i^\mu} = [\widehat{\boldsymbol{s}}]_i), \text{ by independence of the } \widehat{S_i^\mu}.\end{aligned}$$

Secondly, we have:

$$\begin{aligned}\mathbb{P}(\widehat{S_i^\mu} = \widehat{v}) &= \mathbb{P}([\widehat{S^\mu}]_i = \widehat{v}) \\ &= \mathbb{P}(\widehat{S^\mu} \in \{\widehat{\boldsymbol{s}} : [\widehat{\boldsymbol{s}}]_i = \widehat{v}\}) \\ &= \sum_{\widehat{\boldsymbol{s}}:[\widehat{\boldsymbol{s}}]_i=\widehat{v}} \mathbb{P}(\widehat{S^\mu} = \widehat{\boldsymbol{s}}).\end{aligned}$$

□

**Remark 5.2.6.** *In the above corollary we take the "$i^{th}$ compartment of an equivalence class". Intuitively, this is the same as taking any element of that class, taking its $i^{th}$ compartment, then taking the $i^{th}$ class of that compartment. A rigourous statement and proof of this can be found in Section B.2.6.*

In more familiar notation, if we take $\widehat{\boldsymbol{\pi}^\mu}$ to be the $\sim_{\oplus\mu}$ class-aggregated stationary distribution of the controlled whole CTMDP $\mathfrak{C}$ under CS policy $\mu$ , and take $\widehat{\boldsymbol{\pi}^{\mu_i}}$ to be the $\sim_{\oplus\mu_i}$ class-aggregated stationary distribution of the controlled compartment CTMDPs $\mathfrak{C}_i$ under CS policies $\mu_i = [\mu]_i$, then we have $\widehat{\pi^\mu}(\widehat{\boldsymbol{s}}) = \prod_{i=1}^{I}\widehat{\pi^{\mu_i}}([\widehat{\boldsymbol{s}}]_i)$ and $\widehat{\pi^{\mu_i}}(\widehat{v}) = \sum_{\widehat{\boldsymbol{s}}:[\widehat{\boldsymbol{s}}]_i=\widehat{v}}\widehat{\pi^\mu}(\widehat{\boldsymbol{s}})$. We can also write this in terms of decision variables for OP. Let $\mu$ be a CS policy, $\pi(\boldsymbol{s},\boldsymbol{a})$ be the corresponding basic feasible solution to OP for the whole CTMDP $\mathfrak{C}$, and $\pi_i(v,b)$ for $v \in \mathcal{S}_i, b \in \mathcal{A}_i(v)$ be the corresponding basic feasible solution to OP for the $i$th compartment CTMDP $\mathfrak{C}_i$ under policy $[\mu]_i$. Consider the class-aggregated solutions $\widehat{\pi}(\widehat{\boldsymbol{s}},\widehat{\boldsymbol{a}}) := \sum_{(\boldsymbol{s}',\boldsymbol{a}')\in\widehat{\boldsymbol{s}},\widehat{\boldsymbol{a}}}\pi(\boldsymbol{s}',\boldsymbol{a}')$ under $\sim_\oplus$ and $\widehat{\pi}_i(\widehat{v,b}) = \sum_{(v',b')\in\widehat{v,b}}\pi_i(v',b')$ under $\sim_{\oplus_i}$. Immediately from Lemma 5.2.1 and Corollary 5.2.6 we

have:

$$\widehat{\pi}(\widehat{\boldsymbol{s},\boldsymbol{a}}) = \prod_{i=1}^{I} \pi_i([\widehat{\boldsymbol{s},\boldsymbol{a}}]_i), \tag{5.2.8}$$

$$\widehat{\pi}_i(\widehat{v,b}) = \sum_{(v',b')\in\widehat{v,b}} \sum_{(\boldsymbol{s},\boldsymbol{a}):[\boldsymbol{s},\boldsymbol{a}]_i=(v',b')} \pi(\boldsymbol{s},\boldsymbol{a}) = \sum_{\widehat{\boldsymbol{s},\boldsymbol{a}}:[\widehat{\boldsymbol{s},\boldsymbol{a}}]_i=(v',b')} \widehat{\pi}(\widehat{\boldsymbol{s},\boldsymbol{a}}). \tag{5.2.9}$$

The above equations will be useful for the analysis in Section 5.3. This concludes the theoretical analysis of decomposable impulsive CTMDPs controlled by coherent separable policies. What remains is for us to develop a scalable solution methodology for the decomposable framework. The state and action spaces of a decomposable impulsive CTMDP grow exponentially in the number of compartments, implying that this framework suffers from the curse of dimensionality. The next section proposes a solution methodology that leverages their decomposable nature.

## 5.3 Approximate Solution Methodology

In this section we theoretically motivate and outline a solution methodology for decomposable impulsive CTMDPs. In Section 5.3.1 we show that, with additional constraints that enforce a coherent separable policy, the linear program associated with a decomposable impulsive CTMDP is equivalent to a block-diagonal program with one block per compartment CTMDP. In the single-objective case, this offers an obvious decomposition into separate subproblems that collectively form a solution for the whole problem. In Section 5.3.2, we tackle the multi-objective case and leverage the block-diagonal structure by applying Dantzig-Wolfe decomposition. This yields multi-objective subproblems that are easier to solve, and can be pieced back together to form near-Pareto-optimal solutions to the whole problem using a multi-objective integer program. Recognising the potential computational difficulty of this exact methodology, in Section 5.3.4 we propose a relaxation in order to obtain a heuristic solution framework.

### 5.3.1 Block-Diagonalisation

The core requirement of our heuristic is that the decomposable impulsive CTMDP will be controlled using a coherent separable policy. In the terminology of Powell (2022), we call this a *policy function approximation*, or PFA. In order to find the best such policy, we must add additional constraints to the original problem OP. To achieve this, we must introduce binary variables to avoid non-linear constraints. This yields a problem that we call the Coherent Separable Mixed Integer Linear Program, or CS-MILP. Note that this is not a problem formulation that one would implement using a MILP solver, but a theoretical step towards a decomposition-based method.

$$
\begin{aligned}
\text{(CS-MILP)} \qquad & \min_{\boldsymbol{\pi},\boldsymbol{y}} g_1 = \sum_{\boldsymbol{s},\boldsymbol{a}\in\mathcal{SA}} c_1(\boldsymbol{s}\oplus\boldsymbol{a})\pi(\boldsymbol{s},\boldsymbol{a}) \\
& \quad \vdots \\
& \min_{\boldsymbol{\pi},\boldsymbol{y}} g_K = \sum_{\boldsymbol{s},\boldsymbol{a}\in\mathcal{SA}} c_K(\boldsymbol{s}\oplus\boldsymbol{a})\pi(\boldsymbol{s},\boldsymbol{a}) \\
& \text{s.t. } (5.2.3) - (5.2.5) \\
& \quad \pi(\boldsymbol{s},\boldsymbol{a}) \le y(\boldsymbol{s},\boldsymbol{a}) \text{ for all } (\boldsymbol{s},\boldsymbol{a}) \in \mathcal{SA}, && (5.3.1) \\
& \quad \sum_{\boldsymbol{a}'\in\mathcal{A}(\boldsymbol{s}),\boldsymbol{a}'\neq\boldsymbol{a}} \pi(\boldsymbol{s},\boldsymbol{a}') \le 1 - y(\boldsymbol{s},\boldsymbol{a}), \text{ for all } (\boldsymbol{s},\boldsymbol{a}) \in \mathcal{SA}, && (5.3.2) \\
& \quad \sum_{\boldsymbol{a}'\in\mathcal{A}(\boldsymbol{s}),\boldsymbol{a}'\neq 0} \pi(\boldsymbol{s}\oplus\boldsymbol{a},\boldsymbol{a}') \le 1 - y(\boldsymbol{s},\boldsymbol{a}), \text{ for all } (\boldsymbol{s},\boldsymbol{a}) \in \mathcal{SA}, && (5.3.3) \\
& \quad \sum_{\substack{(\boldsymbol{s}',\boldsymbol{a}')\in\mathcal{SA}: \\ [\boldsymbol{s}']_i=[\boldsymbol{s}]_i,[\boldsymbol{a}']_i\neq[\boldsymbol{a}]_i}} \pi(\boldsymbol{s}',\boldsymbol{a}') \le 1 - y(\boldsymbol{s},\boldsymbol{a}), \text{ for all } (\boldsymbol{s},\boldsymbol{a}) \in \mathcal{SA}, && (5.3.4) \\
& \quad y(\boldsymbol{s},\boldsymbol{a}) \in \{0,1\}, \text{ for all } (\boldsymbol{s},\boldsymbol{a}) \in \mathcal{SA}.
\end{aligned}
$$

We associate with each state-action pair a binary variable $y(\boldsymbol{s}, \boldsymbol{a})$ indicating whether or not our policy uses action $\boldsymbol{a}$ in state $\boldsymbol{s}$. Constraint (5.3.1) enforces this relation. Constraint (5.3.2) ensures that the policy is deterministic; that is, if $\pi(\boldsymbol{s}, \boldsymbol{a}) > 0$, then $y(\boldsymbol{s}, \boldsymbol{a}) = 1$, so this constraint forces $\pi(\boldsymbol{s}, \boldsymbol{a}') = 0$ for all $\boldsymbol{a}' \neq \boldsymbol{a}$. Constraint (5.3.3) ensures that the policy is coherent, so if action $\boldsymbol{a}$ is used in a recurrent state $\boldsymbol{s}$ (i.e., $y(\boldsymbol{s}, \boldsymbol{a}) = 1$), then we force $\boldsymbol{a}' = 0$ to be the only allowable action for state $\boldsymbol{s} \oplus \boldsymbol{a}$. Constraint (5.3.4) ensures that the policy is separable, so if action $\boldsymbol{a}$ is used in state $\boldsymbol{s}$, then for all other states $\boldsymbol{s}'$ such that $[\boldsymbol{s}']_i = [\boldsymbol{s}]_i$ we ensure that the allowable actions $\boldsymbol{a}'$ satisfy $[\boldsymbol{a}']_i = [\boldsymbol{a}]_i$.

We wish to "block-diagonalise" CS-MILP by splitting it into $I$ independent CT-MDPs. To do this, we define a new problem that we call the Decomposed Coherent Mixed Integer Program, or DC-MIP. Again, this is not a formulation to be solved

directly, but a theoretical tool.

$$\text{(DC-MIP)} \quad \min_{\boldsymbol{\pi}_i, \boldsymbol{y}_i} g_1 = f_1 \left( \left( \sum_{(s_i,a_i)\in\mathcal{SA}_i} c_{i,1}(s_i \oplus a_i)\pi_i(s_i,a_i) \right)_{i=1}^{I} \right)$$

$$\vdots$$

$$\min_{\boldsymbol{\pi}_i, \boldsymbol{y}_i} g_K = f_K \left( \left( \sum_{(s_i,a_i)\in\mathcal{SA}_i} c_{i,K}(s_i \oplus a_i)\pi_i(s_i,a_i) \right)_{i=1}^{I} \right)$$

$$\text{s.t. for each } i \in \{1,\dots,I\}$$

$$\sum_{(s_i,a_i)\in\mathcal{SA}_i} q_i(s_i,a_i,s_i')\pi_i(s_i,a_i) = 0, \text{ for all } s_i' \in \mathcal{S}_i, \tag{5.3.5}$$

$$\sum_{(s_i,a_i)\in\mathcal{SA}_i} \pi_i(s_i,a_i) = 1, \tag{5.3.6}$$

$$\pi_i(s_i,a_i) \geq 0, \text{ for all } (s_i,a_i) \in \mathcal{SA}_i. \tag{5.3.7}$$

$$\pi_i(s_i,a_i) \leq y_i(s_i,a_i), \text{ for all } (s_i,a_i) \in \mathcal{SA}_i, \tag{5.3.8}$$

$$\sum_{a_i'\in\mathcal{A}_i(s_i),a_i'\neq a_i} \pi_i(s_i,a_i') \leq 1 - y_i(s_i,a_i), \text{ for all } (s_i,a_i) \in \mathcal{SA}_i, \tag{5.3.9}$$

$$\sum_{a_i'\in\mathcal{A}_i(s),a_i'\neq 0} \pi_i(s_i \oplus a_i,a_i') \leq 1 - y_i(s_i,a_i), \text{ for all } (s_i,a_i) \in \mathcal{SA}_i, \tag{5.3.10}$$

$$y_i(s_i,a_i) \in \{0,1\}, \text{ for all } (s_i,a_i) \in \mathcal{SA}_i. \tag{5.3.11}$$

Note that in the objective functions of the DC-MIP, the $\pi_i$ refer to the state-action frequencies of the compartment CTMDPs, and are not defined in terms of the global state-action frequencies as in the definition of the decomposable cost function. For each of the $i$ compartment CTMDPs, constraints (5.3.5) give the balance equations, (5.3.6) and (5.3.7) ensure $\pi_i$ is a valid probability distribution, (5.3.8) ensures $y_i(s_i,a_i)$ is 1 if action $a_i$ is used in state $s_i$ if $s_i$ is recurrent, (5.3.9) ensures that policies are deterministic, (5.3.10) ensures that policies are coherent, and (5.3.11) ensures that the

$y_i$ variables are binary.

We observe that there are no linking constraints between any of the $i$ sub-CTMDPs, so this problem has a strict block-diagonal structure, allowing for an easy Dantzig-Wolfe decomposition. However, before this, it must be shown that the two problems CS-MILP and DC-MIP are in some way equivalent. The sense of equivalence we require is that for any solution in one programme, there exists a solution to the other programme with the same objective values. In order to prove this, we will show that it is possible to move freely between the space of solutions to either programme (i.e., the mixed integer programming perspective) in such a way that objective values are preserved. As this result is somewhat expected, we simply state the theorem here and defer the proof to the appendix.

**Theorem 5.3.1.** *There exists a solution to CS-MILP with objective value $\boldsymbol{g} = \bar{\boldsymbol{g}}$ if and only if there exists a solution to DC-MIP with objective value $\boldsymbol{g} = \bar{\boldsymbol{g}}$.*

*Proof.* See Section B.3. □

As such, further analyses need only consider the decomposed problem DC-MIP, as every solution for CS-MILP has an equivalent solution in DC-MIP and vice versa. In the single-objective case, this completes our analysis, as each block of DC-MIP can be optimised separately alongside its decomposed cost function, the integer variables and additional constraints can be omitted and the block reduces to an instance of OP, which can be solved via LP, value iteration or other methods. This will always result in an optimal DS policy which, if needed, can be sequenced to obtain a coherent policy. The coherent policies together then give the coherent policy for the whole system. However, in the multi-objective case, the task of putting the policies together is non-trivial, as we have a set of Pareto-optimal solutions per compartment, and these must be pieced together in order to obtain non-dominated solutions for the whole problem. The next subsection explores how to do this.

### 5.3.2 Dantzig-Wolfe Decomposition

To aid in the solution of DC-MIP, we can leverage the block-diagonal structure of the problem and apply *Dantzig-Wolfe decomposition*. Under such a decomposition for a continuous linear problem of the form $\min_{\boldsymbol{x}}\{f(\boldsymbol{x}) : A\boldsymbol{x} = \boldsymbol{b}, x \geq 0\}$ with a block-diagonal component, we recognise that each block of the block-diagonal matrix defines a polytope in the decision space. For each block, we can replace the decision variables with *disaggregated* variables $\lambda_s^i \in [0,1]$, each of which corresponds to a corner of the polytope, or equivalently a basic feasible solution of the subproblem corresponding to that block. By convexity of the feasible region, any solution can be recovered by a weighted sum of the polytope corners, with weights summing to one. When considering DC-MIP, due to the binary variables, the polytope is the convex hull of the integral solutions to each block. Each basic feasible solution of that polytope will correspond to a coherent policy $\mu_i$ that induces a unichain CTMC. Some of these solutions may even correspond to the same policy, due to symmetries on the $y$ variables for transient states. We also recognise that for each coherent policy $\mu_i$, we have corresponding long-run average costs $g_{i,k}(\mu_i) = \sum_{s_i \in \mathcal{S}_i} c_{i,k}(s_i \oplus \mu_i(s_i))\pi_i(s_i, \mu_i(s_i))$. We therefore construct the following decomposed coherent master problem, or DC-MP:

$$
\begin{aligned}
\text{(DC-MP)} \quad \min_{\boldsymbol{\lambda}} g_1 &= f_1\left(\left(\sum_{\mu_i \in \mathcal{M}_i^{DC}} g_{i,1}(\mu_i)\lambda_{\mu_i}^i\right)_{i=1}^{I}\right) \\
&\vdots \\
\min_{\boldsymbol{\lambda}} g_K &= f_K\left(\left(\sum_{\mu_i \in \mathcal{M}_i^{DC}} g_{i,K}(\mu_i)\lambda_{\mu_i}^i\right)_{i=1}^{I}\right) \\
&\text{s.t. for every } i \in \{1,\ldots,I\}: \\
&\sum_{\mu_i \in \mathcal{M}_i^{C}} \lambda_{\mu_i}^i = 1, \\
&\lambda_{\mu_i}^i \in \{0,1\}, \text{ for all } i \in [I], \mu_i \in \mathcal{M}_i^C,
\end{aligned}
$$

where $\mathcal{M}_i^C$ is the set of all unichain coherent policies for compartment $i$. We constrain the disaggregated variables $\boldsymbol{\lambda}$ to be binary instead of $[0, 1]$ as we do not have linking constraints to be concerned about, and we want to keep our solutions deterministic. In this initial formulation, the number of binary variables is very large; however, this can be alleviated by identifying a much smaller subset of variables to be included, if it can be guaranteed that the resulting formulation still finds all Pareto-optimal solutions.

**Proposition 5.3.1.** *Let $\mathfrak{C}$ be a decomposable impulsive CTMDP with compartments $\mathfrak{C}_i$. A solution to DC-MP is Pareto-optimal only if the compartment policies $\mu_i$ are Pareto-optimal with respect to the vector objective functions $\boldsymbol{g}_i$. If a policy $\mu = (\mu_1, \ldots, \mu_I)$ has sub-policies $\mu_i$ that are not Pareto-optimal, then there exists a solution $\mu'$ with Pareto-optimal sub-policies such that the objective values of $\mu$ are bounded below by $\mu'$.*

*Proof.* Construct $\mu'$ as follows: let $\mu_i' = \mu_i$ if $\mu_i$ is Pareto-optimal; otherwise, choose a Pareto-optimal $\mu_i'$ that dominates $\mu_i$. Now $\boldsymbol{g}_i(\mu_i') \leq \boldsymbol{g}_i(\mu_i)$ component-wise, and as the $f_k$ are increasing functions, $f_k(g_{1,k}(\mu_1'), \ldots, g_{I,k}(\mu_I')) \leq f_k(g_{1,k}(\mu_1), \ldots, g_{I,k}(\mu_I))$ for all $k$. □

By Proposition 5.3.1, we can reduce the number of decision variables by including only the ones that correspond to a Pareto-optimal compartment policy. We write $\mathrm{PF}(\mathcal{M}_i^C)$ to be the set of Pareto-optimal polices in $\mathcal{M}_i^C$ with respect to $\boldsymbol{g}_i$, i.e., the Pareto front. As such we can write the reduced master problem, which we call DC-

RMP:

$$
\begin{aligned}
\text{(DC-RMP)} \quad & \min_{\boldsymbol{\lambda}} g_1 = f_1 \left( \left( \sum_{\mu_i \in \mathrm{PF}(\mathcal{M}_i^{DC})} g_{i,1}(\mu_i) \lambda^i_{\mu_i} \right)_{i=1}^{I} \right) \\
& \quad \vdots \\
& \min_{\boldsymbol{\lambda}} g_K = f_K \left( \left( \sum_{\mu_i \in \mathrm{PF}(\mathcal{M}_i^{DC})} g_{i,K}(\mu_i) \lambda^i_{\mu_i} \right)_{i=1}^{I} \right) \\
& \text{s.t. for every } 1 \leq i \leq I : \\
& \quad \sum_{\mu_i \in \mathrm{PF}(\mathcal{M}_i^{C})} \lambda^i_{\mu_i} = 1, \\
& \quad \lambda^i_{\mu_i} \in \{0, 1\}, \text{ for all } \mu_i \in \mathrm{PF}(\mathcal{M}_i^{C}).
\end{aligned}
$$

By Proposition 5.3.1, any Pareto-optimal solution to DC-MP has an equivalent in DC-RMP, and vice versa. If $\mathrm{PF}(\mathcal{M}_i^{DC})$ can be obtained exactly, then we can obtain Pareto-optimal solutions for DC-RMP and in turn for CS-MILP using this decomposition. However, in practice, finding $\mathrm{PF}(\mathcal{M}_i^{DC})$ exactly is difficult, as it refers to the set of DS policies that are Pareto-optimal only with respect to the other DS policies. This means the set contains policies that are not optimal for any mixed-objective scalarisation of the objective function, and are therefore hard to find. As such, we instead consider subsets $\overline{\mathrm{PF}}(\mathcal{M}_i^{DC})$ of the Pareto front which are representative of the front, but which are easier to find.

As currently presented, DC-RMP may be a non-linear MIP due to the $f_k$ function, making it hard to solve. Ideally, we would want an MILP formulation to be solved by an off-the-shelf MILP solver. For the decomposable cost functions discussed in this chapter, the following subsection discusses some linearisations.

### 5.3.3 Linearisations

We can linearise DC-RMP if we can apply a strictly increasing transformation $T_k$ to each $f_k$ such that $(T_k \circ f_k)(\dots, \sum_{\mu_i \in \mathrm{PF}(\mathcal{M}_i^{DC})} g_{i,k}\lambda^i_{\mu_i}, \dots)$ is a linear combination of the $\lambda^i_{\mu_i}$ whenever the $\lambda^i_{\mu_i}$ satisfy the constraints, as we can then use any MILP solver to solve DC-RMP. This is because, in general for a multi-objective optimisation problem with decision variables $\boldsymbol{x} \in \mathcal{X}$, a solution $\boldsymbol{x}$ is Pareto-optimal for objectives $(f_1, \dots, f_K)$ if and only if it is also Pareto-optimal for $(T_1 \circ f_1, \dots, T_K \circ f_k)$ where the $T_k$ are strictly increasing functions. Such transformations exist for all additive, multiplicative, and logical-OR cost functions. Of course, this is trivially evident for the additive costs case, with $T_k(x) = x$. In the case of multiplicative costs, we choose $T_k(x) = \ln(x)$ and recall $f_k(\boldsymbol{x}) = \prod_{i=1}^{I} x_i$, and find:

$$
\begin{aligned}
(T_k \circ f_k)\left(\left(\sum_{\mu_i \in \mathrm{PF}(\mathcal{M}_i^C)} g_{i,k}(\mu_i)\lambda^i_{\mu_i}\right)_{i=1}^{I}\right) &= \ln\left\{\prod_{i=1}^{I}\sum_{\mu_i \in \mathrm{PF}(\mathcal{M}_i^{DC})} g_{i,k}(\mu_i)\lambda^i_{\mu_i}\right\} \\
&= \sum_{i=1}^{I} \ln\left\{\sum_{\mu_i \in \mathrm{PF}(\mathcal{M}_i^C)} g_{i,k}(\mu_i)\lambda^i_{\mu_i}\right\} \\
&= \sum_{i=1}^{I}\sum_{\mu_i \in \mathrm{PF}(\mathcal{M}_i^C)} \ln\left\{g_{i,k}(\mu_i)\right\}\lambda^i_{\mu_i},
\end{aligned}
$$

where the last equality holds only for binary $\lambda^i_{\mu_i}$ such that $\sum_{\mu_i \in \mathrm{PF}(\mathcal{M}_i^C)} \lambda^i_{\mu_i} = 1$. Similarly, for the logical-OR case, we choose $T_k(x) = -\ln(1-x)$ and recall $f(\boldsymbol{x}) =$

$1 - \prod_{i=1}^{I}(1 - x_i)$ and find:

$$
\begin{aligned}
&(T_k \circ f_k)\left(\left(\sum_{\mu_i \in \mathrm{PF}(\mathcal{M}_i^C)} g_{i,k}(\mu_i)\lambda_{\mu_i}^i\right)_{i=1}^{I}\right) \\
&= -\ln\left\{1 - \left(1 - \prod_{i=1}^{I}\left(1 - \sum_{\mu_i \in \mathrm{PF}(\mathcal{M}_i^C)} g_{i,k}(\mu_i)\lambda_{\mu_i}^i\right)\right)\right\} \\
&= -\sum_{i=1}^{I} \ln\left\{1 - \sum_{\mu_i \in \mathrm{PF}(\mathcal{M}_i^C)} g_{i,k}(\mu_i)\lambda_{\mu_i}^i\right\} \\
&= -\sum_{i=1}^{I} \sum_{\mu_i \in \mathrm{PF}(\mathcal{M}_i^C)} \ln\left\{1 - g_{i,k}(\mu_i)\right\}\lambda_{\mu_i}^i,
\end{aligned}
$$

where again the equality holds only for suitable $\lambda_{\mu_i}^i$. Therefore, if all objectives are additive, multiplicative, or logical-OR cost functions, DC-RMP can be reformulated as having linear objective functions, and is therefore a mixed integer linear program that can be solved by any MILP solver.

### 5.3.4 Solution Framework

As it stands, obtaining the full Pareto fronts $\mathrm{PF}(\mathcal{M}_i^C)$ for the subproblems and plugging them into DC-RMP provides an exact solution method for CS-MILP and DC-MIP. However, it does not necessarily follow that such solutions are Pareto-optimal for the original problem OP, as we have made the simplifying assumption throughout that we limit ourselves to separable policies. There are also some remaining problems with this approach. Obtaining $\mathrm{PF}(\mathcal{M}_i^C)$ can be challenging in its own right even for small CTMDPs, and the set itself can still have a high cardinality and result in an instance of DC-RMP with a large number of binary variables. This is especially true if there's a large number of objectives, leading to more policies being Pareto-optimal. Also, in general, even single-objective MDPs can be difficult or impossible to solve exactly due to the curse of dimensionality.

To convert our exact methodology into a flexible, scalable approximate methodology, we may simply replace the exact Pareto fronts $\mathrm{PF}(\mathcal{M}_i^C)$ in DC-RMP with *approximate* Pareto fronts $\overline{\mathrm{PF}}(\mathcal{M}_i^C)$, obtaining DC-RMP-Approx. There are two main ways that the solution sets $\overline{\mathrm{PF}}$ are approximate: they may be incomplete (i.e., missing Pareto-optimal solutions), or the solutions present may not be Pareto-optimal. Nevertheless, if the solution set is reasonably representative of the objective space (i.e., captures the different balances between the different objectives), and the extent of sub-optimality is acceptable, this method may still be worthwhile in practice. We now present our solution methodology as a meta-algorithm:

1. Choose a decomposition $\{\mathfrak{C}_i\}_{i=1}^I$ of the decomposable impulsive CTMDP $\mathfrak{C}$, and identify the compartment cost functions $c_{i,k}$ for each compartment and objective.

2. Obtain an exact or approximate Pareto front of solutions to each compartment subproblem, taking care to ensure all policies are coherent. This may be done with exact methods by scalarising the problem (e.g. using the weighted-sum) and then using methods such as value iteration or linear programming, or by using approximate methods such as multi-objective reinforcement learning (MORL) (Van Moffaert and Nowé, 2014) or bespoke heuristics. Where non-coherent policies are found, apply sequencing to obtain coherent policies (but recall that this may result in further sub-optimality when the non-coherent policies are sub-optimal).

3. Use the exact fronts or approximate fronts as the subproblem solution sets in DC-RMP or DC-RMP-Approx (respectively), and use a scalarising method together with a MILP solver (or some heuristic solution method) to obtain the Pareto fronts for these problems.

This meta-algorithm allows for a lot of flexibility, including the choice of decomposition, the solution methods for the subproblems, and the handling of the multiple objectives in the reduced master problem.

It remains for us to explain why one might use this solution methodology over other popular approximate methodologies that could be applied directly to decomposable impulsive CTMDPs, such as MORL. In the case where compartment MDPs are small enough to be solved exactly, this method benefits from the true optimality of the subproblems, and the lack of complications that arise from reinforcement learning methods (such as the choice of value function approximation or neural network architecture, parameter tuning, etc). On the other hand, in the case where compartment MDPs cannot be solved exactly, we argue that it may be easier to apply MORL methods or design bespoke heuristics for the simpler compartment CTMDPs than for the whole problem. In either case, our method can also serve as a benchmark or as a warm start for reinforcement learning methods. For example, we could provide policies from our approximate Pareto front as the baseline policies for methods such as approximate policy iteration or rollout (Bertsekas, 2021), or as the initial "actor" in an actor-critic method (Konda and Tsitsiklis, 1999).

In this section, we have developed a flexible decomposition-based heuristic solution methodology that can be used to approximately solve decomposable impulsive CTMDPs with multiple objectives, and achieved this using Dantzig-Wolfe decomposition. In the next section, we consider an application of the decomposable impulsive CTMDP framework, and demonstrate the proposed solution methodology.

## 5.4 Application

In this section we demonstrate an application of our methodology to a bi-objective dynamic maintenance problem, centred on a series-parallel system. Such a system is comprised of multiple subsystems connected in series, with each subsystem made up of redundant components connected in parallel. A subsystem is considered functional if at least one of its components is functional, but the whole system is considered

functional only if all of the subsystems are functional. We wish to find Pareto optimal solutions with respect to maintenance costs and reliability. This problem can be thought of as a natural extension of the redundancy allocation problem (RAP), which is an NP-hard non-linear combinatorial optimisation problem arising from reliability engineering, where redundant copies of components can be added to a system and the system only fails when all copies of a critical component are damaged (Chern, 1992). Of particular relevance to this study are repairable series-parallel systems that make use of Markovian modelling to model system degradation and repairs. Kayedpour et al. (2017) and Tavana et al. (2018) both make use of CTMCs in modelling the degradation and repair processes of series-parallel systems, where the former uses binary state modelling (healthy or damaged, with nothing in between), and the latter uses multi-state modelling, allowing for a more granular scale of degradation states. Both models work under the assumption that components are always repaired immediately. We extend the binary state model by relaxing the assumption that components are always repaired and instead suppose that decisions about repairs are made as part of a bi-objective sequential decision making problem under uncertainty. This yields a bi-objective CTMDP, in which it is natural to think of the subsystems with parallel redundancy as decomposable compartments.

In Section 5.4.1 we formulate our problem by first defining the compartment CTMDPs, then taking the direct product of these and choosing the cost functions. In Section 5.4.2 we discuss the specific implementation of our methodology to this problem. Section 5.4.3 then provides a comparison between an exact method and our heuristic decomposition-based method in terms of runtime, optimality, and number of solutions in the Pareto fronts. We also apply our methodology to larger instances that cannot be solved exactly in order to demonstrate the scalability of our solution methodology.

### 5.4.1 Problem Definition

We construct our CTMDP using the direct product method, which begins by defining the structures of the compartment CTMDPs with components connected in parallel and then uses these to construct the whole series-parallel CTMDP. Recall that a cost-free impulsive CTMDP is defined as a tuple $\mathfrak{C}_i = \langle \mathcal{S}_i, (\mathcal{A}_i(s_i))_{s_i \in \mathcal{S}_i}, q^+ : \mathcal{S}_i \times \mathcal{S}_i \to \mathbb{R}_{\geq 0}, \oplus_i : \mathcal{SA}_i \to \mathcal{S}_i \rangle$. We assume that the components in a subsystem are homogeneous, so there is no need to differentiate between specific components. We assume that each component can be in one of three conditions: healthy, damaged, or repairing. A healthy component is one that can be used, a damaged component cannot be used and is not undergoing maintenance, and a repairing component cannot be used but is undergoing maintenance. Including a repairing condition in the state is based on the assumption that after a repair has been started, it must be seen through to completion (or fail). Without this inclusion, a repair that is started in one state could be discontinued after a transition. Due to the homogeneity of components within a subsystem, it is sufficient for the state to include the numbers of components that are damaged and repairing, rather than detailing specifically which copies are in which condition.

Suppose that a subsystem $i$ has $M_i$ components. The sub-state must give the number of copies of the component that are currently repairing and the number damaged. The number that are healthy can then be inferred from the total number of components $M_i$. We define $\mathcal{S}_i := \{(s_{i,1}, s_{i,2}) \in \mathbb{Z}^2_{\geq 0} : s_{i,1} + s_{i,2} \leq M_i\}$, which gives the set of all possible combinations of the number of components in subsystem $i$ that are repairing ($s_{i,1}$) or damaged ($s_{i,2}$). For notational convenience, we write $s_{i,0} := M_i - s_{i,1} - s_{i,2}$ as the number of healthy components in subsystem $i$. For the action spaces, we define: $\mathcal{A}_i(s_i) := \{a \in \mathbb{Z} : 0 \leq a \leq s_{i,2}\}$, so the number of components in subsystem $i$ that can be put into repair is bounded above by the number of damaged components in that subsystem. For the impulsive operator, we define $\oplus_i : \mathcal{SA}_i \to \mathcal{S}_i$ as $\boldsymbol{s}_i \oplus_i \boldsymbol{a}_i = (s_{i,1} + a_i, s_{i,2} - a_i)$. Finally, we must define the positive transition rates $q_i^+$. There are

two types of event which cause a state transition:

- **Degradation** - The degradation of a component due to an exogenous random event. The interarrival times of such events are assumed to be independent and exponentially distributed with rate $\alpha_i > 0$ for subsystem $i$. Note that we assume that degradations can occur even while a component is being repaired.

- **Successful Repair** - The event that changes the condition of a repairing component to 'healthy'. We assume that repair completion times are independent and exponentially distributed with rate $\tau_i > 0$ for subsystem $i$.

We can then construct the positive transition rates $q^+$:

$$q_i^+(\boldsymbol{s}_i, \boldsymbol{s}_i') = \begin{cases} s_{i,0}\alpha_i, & \boldsymbol{s}_i' = \boldsymbol{s}_i + \boldsymbol{e}_{i,2}, \\ s_{i,1}\tau_i, & \boldsymbol{s}_i' = \boldsymbol{s}_i - \boldsymbol{e}_{i,1}, \\ 0, & \text{otherwise}, \end{cases}$$

where $\boldsymbol{e}_{i,k}$ is a 2-tuple with $\boldsymbol{e}_{i,1} = (1, 0)$ and $\boldsymbol{e}_{i,2} = (0, 1)$. The first line gives the rate of a healthy component in subsystem $i$ failing and becoming damaged and the second line gives the rate of a repairing component in subsystem $i$ completing its repair and becoming healthy.

We can also construct the vector costs of the compartment CTMDPs. The first cost reflects the resources required by ongoing repairs. If a component in subsystem $i$ is currently under repair then we incur a cost at rate $r_i$ per unit time. Our second cost is an indicator function for system failure, where no healthy components are available. Symbolically, we have an operational cost rate $c_i^o(\boldsymbol{s}_i) = s_{i1}r_i$ and a failure indicator cost rate $c_i^f(\boldsymbol{s}_i) = \mathbb{I}\{s_{i0} = 0\}$.

To create the cost-free version of the series-parallel model, under the assumption that no event in one subsystem would cause an event or change in another, we take the direct product of the subsystems $\mathfrak{C}_i$ and let $\mathfrak{C} = \bigotimes_{i=1}^{I} \mathfrak{C}_i$ denote the cost-free

decomposable impulsive CTMDP for a series-parallel system. The operational cost of the system is defined as the sum of the operational costs of the subsystems, and hence $c^o(\boldsymbol{s}) = \sum_{i=1}^{I} c_o^i([\boldsymbol{s}]_i)$. Clearly, this cost is decomposable, as per Proposition 5.2.5. The cost of system failure is slightly different, because a series-parallel system fails whenever at least one subsystem fails. As such, we have $c^f(\boldsymbol{s}) = \mathbb{I}\{\bigvee_{i=1}^{I} c_i^f([\boldsymbol{s}]_i) = 1\}$, which is decomposable by Proposition 5.2.5 because it is a logical-OR cost function.

### 5.4.2 Methodology

The aim of our approach is to find a well-populated set of Pareto-optimal, or approximately Pareto-optimal, solutions for our dynamic maintenance problem. This does not necessarily mean finding the entire Pareto front, as this can be computationally very expensive, hard to verify, and can result in a large number of very similar solutions.

#### Exact Method

The first approach, and our point of comparison for the decomposition-based heuristic, is an exact method based on the original problem OP. We scalarise the problem by using the mixed-objectives method, and consider a parameterised single objective of the form

$$g = \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}} \left(c^o(\boldsymbol{s},\boldsymbol{a}) + Pc^f(\boldsymbol{s},\boldsymbol{a})\right).$$

The parameter $P$ acts as a penalty for system failure. We expect low values of $P$ to result in more passive maintenance policies, and high values to result in more active policies. A sufficiently high value of $P$ should yield the policy that always repairs all components as soon as they become damaged. As this is a mixed-objective problem with positive objective weights, solutions to this for any value of $P$ will be Pareto-optimal.

We call the resulting problem the scalarised original problem, or S-OP.

$$
\begin{aligned}
\text{(S-OP)} \qquad & \min_{\boldsymbol{\pi}} g = \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}} \left(c^o(\boldsymbol{s},\boldsymbol{a}) + Pc^f(\boldsymbol{s},\boldsymbol{a})\right)\pi(\boldsymbol{s},\boldsymbol{a}) \\
& \text{s.t.} \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}} q(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}')\pi(\boldsymbol{s},\boldsymbol{a}) = 0, \text{ for all } \boldsymbol{s}' \in \mathcal{S}, \\
& \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}} \pi(\boldsymbol{s},\boldsymbol{a}) = 1, \\
& \pi(\boldsymbol{s},\boldsymbol{a}) \geq 0, \text{ for all } (\boldsymbol{s},\boldsymbol{a}) \in \mathcal{SA}.
\end{aligned}
$$

The above problem can be solved for any value of $P$ using an LP solver. Preliminary experiments using Gurobi have shown that whether primal or dual simplex solves the problem fastest depends on the problem instance. In theory, primal simplex is often a good choice for multi-objective optimisation, as we may have a good starting solution obtained from a different objective weighting, and therefore only require a few iterations of primal simplex to re-optimise the problem. However, there are still plenty of cases where dual simplex solves the problem faster from scratch. As such, we use the "deterministic concurrent solver", which runs the primal and dual simplex methods on two different threads, and terminates when one of the algorithms finds an optimal solution. The only thing left to decide is which values of $P$ to consider. The truly exact way to determine the values of $P$ would be to adapt the bi-objective parametric simplex method of Ehrgott (2005) or the dichotomic method of Aneja and Nair (1979). However, as we will demonstrate in Section 5.4.3, problem instances can have an abundance of solutions that are very similar in terms of the two objectives whilst still technically being distinct, so such exact methods would waste a lot of time obtaining and storing very similar solutions, and would become prone to numerical error, especially due to the probabilistic nature of our decision variables. As such, we instead choose to solve for a pre-determined range of $P$ values, in ascending order. To aid computational efficiency, whenever we want to solve the problem for $P_n$ with $n > 1$, we give the primal simplex

a warm start by providing the solution from $P_{n-1}$ as a starting solution, so that the problem doesn't need to be re-solved from scratch. In cases where the increase from $P_{n-1}$ to $P_n$ doesn't result in a new solution, the primal simplex solver can quickly verify that the incumbent solution is still optimal by finding that there are no columns with reduced cost. However, especially for some easier problems, the dual simplex solver may solve it from scratch faster anyway. Additionally, the variable and constraint set need only be constructed once, and not re-built for each value of $P_n$. MILP solvers such as Gurobi allow such behaviour, where variables, constraints, and objectives can all be wrapped in a model, and the objective function can be updated without rebuilding the whole model.

**Heuristic Method**

The second approach is to use our decomposition-based heuristic. To do this, we must obtain an approximate Pareto front $\overline{\mathrm{PF}}(\mathcal{M}_i^{DC})$ for each of the sub-CTMDPs $\mathfrak{C}_i$. We can do this by following the exact method described above, scalarising the bi-objective problem using a penalty parameter $P$ and re-solving the problem for a range of values. We find that sequences of the form $P_n = b^n$ for some $b > 1$ with some maximum number of iterations $n_{\max}$ strike a good balance between the number of solutions, and variety across the Pareto front. This is because we expect the second objective (i.e., failure probability) to vary by orders of magnitude; thus, the penalty parameter should also vary by orders of magnitude. Again, we take care not to needlessly rebuild the model for each value of $P_n$, and sequentially provide previous solutions as starting solutions whenever the objective function is updated. However, we let Gurobi decide which algorithm to use (and as such whether to use the provided starting solution) automatically in this case, as the LPs are small enough that algorithm choice has little impact on runtimes. For each solution, we also need to check that the policy is coherent. If it is not, we must determine and evaluate the coherent sequenced version of the

solution. To evaluate a fixed policy, we can use a version of S-OP with only one action permitted per state. This is equivalent to solving the equation $\boldsymbol{\pi}^T Q = 0$; however, we continue to use Gurobi for this, rather than a Linear Algebra library, due to its high precision. The solutions obtained from this method can be recombined via the reduced master problem DC-RMP. After applying the suitable linearising transformation to the second objective function, we obtain the following bi-objective approximate reduced master problem, which we call BO-ARMP:

$$
\begin{aligned}
\text{(BO-ARMP)} \quad & \min_{\boldsymbol{\lambda}} g^o = \sum_{i=1}^{I} \sum_{\mu_i \in \overline{\mathrm{PF}}(\mathcal{M}_i^{DC})} g_i^o(\mu_i) \lambda_{\mu_i}^i \\
& \min_{\boldsymbol{\lambda}} T(g^f) = \sum_{i=1}^{I} \sum_{\mu_i \in \overline{\mathrm{PF}}(\mathcal{M}_i^{DC})} -\ln\left\{1 - g_i^f(\mu_i)\right\} \lambda_{\mu_i}^i \\
& \text{s.t. for every } 1 \le i \le I : \\
& \qquad \sum_{\mu_i \in \overline{\mathrm{PF}}(\mathcal{M}_i^{DC})} \lambda_{\mu_i}^i = 1, \\
& \qquad \lambda_{\mu_i}^i \in \{0,1\}, \text{ for all } \mu_i \in \overline{\mathrm{PF}}(\mathcal{M}_i^{DC}).
\end{aligned}
$$

To handle the bi-objective optimisation, we will use the $\varepsilon-$constraint method, and turn the second objective into a constraint. This yields the $\varepsilon-$constrained approximate reduced master problem, or $\varepsilon-$ARMP:

$$
\begin{aligned}
\text{($\varepsilon$-ARMP)} \quad & \min_{\boldsymbol{\lambda}} g^o = \sum_{i=1}^{I} \sum_{\mu_i \in \overline{\mathrm{PF}}(\mathcal{M}_i^{DC})} g_i^o(\mu_i) \lambda_{\mu_i}^i \\
& \text{s.t. for every } 1 \le i \le I : \\
& \qquad \sum_{\mu_i \in \overline{\mathrm{PF}}(\mathcal{M}_i^{DC})} \lambda_{\mu_i}^i = 1, \\
& \qquad \lambda_{\mu_i}^i \in \{0,1\} \text{ for all } \mu_i \in \overline{\mathrm{PF}}(\mathcal{M}_i^{DC}), \\
& \qquad \sum_{i=1}^{I} \sum_{\mu_i \in \overline{\mathrm{PF}}(\mathcal{M}_i^{DC})} -\ln\left\{1 - g_i^f(\mu_i)\right\} \lambda_{\mu_i}^i \le \varepsilon.
\end{aligned}
$$

This yields a scalarised problem which can be solved via any MILP solver. As in the case of the penalty values $P$, we now need to choose values of $\varepsilon$ to be optimised against. We first choose a starting value for the target probability of failure $g^f_{start}$, then convert this via the linearising transformation, so $\varepsilon_{start} = -\ln(1 - g^f_{start})$. Solving the problem with $\varepsilon = \varepsilon_{start}$ gives a solution that we call $\lambda^{(1)}$, which has second objective value $T(g^{f(1)}) \leq \varepsilon_{start}$. Using a parameter $0 < \gamma < 1$, we can set $\varepsilon = \gamma T(g^{f(1)})$ and use this value of $\varepsilon$ to "cut off" the previous solution, forcing the programme to find a solution with a better second objective value. Solving this programme gives a solution $\lambda^{(2)}$ with second objective value $T(g^{f(2)})$, and then the process repeats. This continues until the algorithm either finds the "always repair" solution or the programme becomes infeasible.

### 5.4.3 Computational Results

We consider problem instances derived from subsets of the solution to the 14-subsystem redundancy allocation problem considered by Fyffe et al. (1968). Each component has a long-run reliability value $p_i$, interpreted as the proportion of time that the component is healthy, assuming it is always repaired immediately. However, repair rates, degradation rates and repair costs are not included in the problem considered by the authors. For the sake of simplicity, we assume that all components have the same repair rate $\tau = 1$ and the same repair cost rate $r = 100$. Given the reliability values $p_i$ for each component, we then choose the degradation rate $\alpha_i$ such that $\tau_i/(\tau_i + \alpha_i) = p_i$, where $\tau_i/(\tau_i + \alpha_i)$ is the long-run proportion of up-time for a component that is always repaired as soon as it becomes damaged. All LPs are solved using Gurobi 10.0.2 (Gurobi Optimization, LLC, 2024), and all code is written in Julia 1.9.3. The code is run on a computer with an Intel Xeon Gold 6348 with four cores clocked at 2.6 GHz and 16 GB of RAM. We obtain subproblems by taking contiguous ranges of subsystems from these 14, and treating them as one system. For example, the problem we call 1:4 is a system comprised of

| Subsystem | $p$ | $\tau$ | r | Copies |
|---|---|---|---|---|
| 1 | 0.91 | 1.0 | 100.0 | 3 |
| 2 | 0.95 | 1.0 | 100.0 | 2 |
| 3 | 0.92 | 1.0 | 100.0 | 3 |
| 4 | 0.85 | 1.0 | 100.0 | 3 |
| 5 | 0.93 | 1.0 | 100.0 | 3 |
| 6 | 0.98 | 1.0 | 100.0 | 2 |
| 7 | 0.91 | 1.0 | 100.0 | 2 |
| 8 | 0.81 | 1.0 | 100.0 | 4 |
| 9 | 0.96 | 1.0 | 100.0 | 2 |
| 10 | 0.85 | 1.0 | 100.0 | 3 |
| 11 | 0.94 | 1.0 | 100.0 | 2 |
| 12 | 0.79 | 1.0 | 100.0 | 4 |
| 13 | 0.99 | 1.0 | 100.0 | 2 |
| 14 | 0.95 | 1.0 | 100.0 | 2 |

Table 5.4.1: Subsystem designs and parameters

subsystems 1, 2, 3 and 4. For the exact method, we use penalty values $P_n = n$ for $n = 1, 2, \ldots, 10000$. For the heuristic method, we use penalty values $P_n = 10^{n/10}$ for $n = 1, 2, \ldots, n_{\max} = 50$. We have found that it is only possible to solve instances exactly if there are no more than four subsystems. All 5-subsystem instances either ran into memory issues whilst building the model and were unable to begin solving, or did not find any solutions except for the "never repair" solution in reasonable time. Specifically, for those 5-subsystem instances that could fit in memory, no other solutions were found within 24 hours.

In order to compare the exact (albeit incomplete) Pareto front with the approximate Pareto front found by our method, in Table 5.4.2 we report summary statistics of an error measurement that we call the Relative Euclidean Pareto Optimality Gap, or REPOG. If a vector $\boldsymbol{z}^*$ is a Pareto-optimal point in $K$-dimensional objective space and $\boldsymbol{z}$ is a point dominated by $\boldsymbol{z}^*$, then the objective-wise relative optimality gaps (ROGs)

are given by the vector:

$$\mathrm{ROG}(\boldsymbol{z}|\boldsymbol{z}^*) = \begin{cases} \left( \left| \frac{z_1 - z_1^*}{z_1^*} \right|, \ldots, \left| \frac{z_K - z_K^*}{z_K^*} \right| \right), & \boldsymbol{z}^* \text{ dominates } \boldsymbol{z}, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

REPOG is then simply given by the Euclidean norm of ROG, so:

$$\mathrm{REPOG}(\boldsymbol{z}|\boldsymbol{z}^*) = ||\mathrm{ROG}((\boldsymbol{z}|\boldsymbol{z}^*)||.$$

Note that in cases where $\boldsymbol{z}$ is equal to $\boldsymbol{z}^*$ in all but one dimension, or where the dimension of the objective space is 1, then REPOG reduces to the standard relative optimality gap. More generally, given a set of points $\mathcal{Z}^*$ all of which dominate $\boldsymbol{z}$, we can define:

$$\mathrm{REPOG}(\boldsymbol{z}|\mathcal{Z}^*) = \min_{\boldsymbol{z}^* \in \mathcal{Z}^*} \mathrm{REPOG}(\boldsymbol{z}|\boldsymbol{z}^*).$$

If the set $\mathcal{Z}^*$ gives the set of all points in the Pareto-front that dominate $\boldsymbol{z}$, then REPOG can be interpreted as the relative distance of a sub-optimal point from the Pareto front.

Given a Pareto front PF which contains only efficient points, but which may not be complete and is finite, and an approximate finite Pareto front $\overline{\mathrm{PF}}$, it can be useful to somehow characterize the sub-optimality of the approximate front against the exact front. One possible method is to simply count the number of points in the approximate front which are dominated by the exact front. A robust and conservative measure of suboptimality is to report the maximum REPOG value amongst the points in the approximate front. We can also report the mean of the non-zero REPOG values. Together, the number of sub-optimal points, the maximum REPOG, and the mean non-zero REPOG give an insight into how well the approximate front compares to the exact front. For example, a high number of sub-optimal points with a low mean and maximum REPOG might suggest a consistently low optimality gap, whereas a low number of sub-optimal points with a high mean and even higher maximum REPOG

may suggest that the approximate front is generally close to optimality, despite having some significant weak spots. We argue that this type of reporting is more useful than the standard hypervolume metric (Shang et al., 2020), which relies on the choice of a reference point, is unintuitive, and obfuscates detail.

From Table 5.4.2 we observe that the exact solution times of the problem instances grow by orders of magnitude with the number of subsystems, whereas the heuristic solution times are all less than 0.5 seconds. On the other hand, the exact method consistently finds many more solutions than the heuristic. This is to be expected, as the heuristic is limited to separable policies. An intuitive explanation is that, although the exact solutions use deterministic policies, one can imagine the policies being "separable" but using the states of other subsystems as sources of randomness; therefore, they can "emulate" randomised separable policies on each subsystem by using the states of other subsystems as a source of pseudo-randomness. This could allow them to strike a finer balance between the two objectives. This may not be the only explanation, but it does provide some insight. Note that in the exact case we report the number of solutions and the number "significant" solutions. The latter refers to the size of a subset of the solutions where there is more than a 0.1% difference in both objective values.

In terms of error, we note that a small handful of instances (1:2, 2:3, 7:8, 12:13, 13:14) achieved a maximum REPOG value of zero or significantly less than 0.01%. We note that all of these instances contain only two subsystems. Problem 10:13 exhibits the largest number of sub-optimal solutions, as well as the highest proportion of solutions being sub-optimal (40%). However, as suggested by the low mean REPOG values and inspection by eye in Figure 5.4.2, we find that the exact and approximate Pareto fronts are quite similar, and that the extent to which the sub-optimal solutions are dominated is minor. The largest maximum REPOG value is 12.4%, given by Problem 1:3. This particular point is the orange point with an operational cost of about 45 in the respective plot in Figure 5.4.1, and is strongly dominated by an exact solution.

However, this particular point is clearly a poor trade-off between two other heuristic solutions on either side, and could be safely ignored by decision-makers.

For instances with five subsystems or more, the exact method becomes computationally infeasible and we are unable to provide comparisons between the heuristic method and optimal solutions. However, we can report the run-times and the numbers of solutions found. The full 14-subsystem problem was solved in 5.22 seconds, with 258 solutions in the approximate Pareto front. All other instances were solved in less time than this, as is to be expected. The full table of results can be found in Section B.4. Focusing on the full 14-subsystem example, Figure 5.4.3 shows the approximate Pareto front for the whole system. It can be seen that the 258 solutions are well-spaced over the Pareto front, and that even if the exact front has more solutions, this will not realistically benefit the decision maker in terms of flexibility. The most reliable solution has an operational cost of 372.25, and a log failure rate (LFR) of -3.472, i.e., a failure rate of 0.031. The Pareto front allows us to do some sensitivity analysis, and weigh the benefits of small sacrifices in reliability against the savings in operational costs. For example, if our target failure rate is 0.04 (LFR $\approx 3.22$), we can select a solution with a cost of 335.48 and a failure rate of 0.0398, providing a substantial saving. As another example, if we have a target failure rate of 0.05 (LFR $\approx -3$), we can choose a solution with a cost of 311.42 and a failure rate of 0.049. Of course, it is not possible to know whether or not these solutions are truly Pareto optimal, except for the two that minimise the failure rate and cost rate. However, it is ultimately up to the decision-maker to decide whether or not the trade-offs are worthwhile.

## 5.5 Conclusion

This chapter has proposed a simpler alternative to the standard approach for the inclusion of impulsive control in CTMDP models found in the literature, and developed

| | Exact | | | Heuristic | | | | |
|---|---|---|---|---|---|---|---|---|
| Problem | Time | Solutions | Significant Solutions | Time | Solutions | Sub-Optimal Solutions | Max REPOG (%) | Mean REPOG (%) |
| 01:02 | 14.74 | 28 | 21 | 0.03 | 9 | 0 | 0.00 | 0.00 |
| 01:03 | 365.15 | 152 | 63 | 0.11 | 24 | 3 | 12.4 | 4.87 |
| 01:04 | 13609.51 | 739 | 183 | 0.37 | 51 | 14 | 6.96 | 1.73 |
| 02:03 | 14.41 | 25 | 18 | 0.03 | 11 | 1 | $< 0.01$ | $< 0.01$ |
| 02:04 | 365.49 | 180 | 88 | 0.12 | 28 | 8 | 2.39 | 1.57 |
| 02:05 | 11030.04 | 727 | 173 | 0.38 | 55 | 21 | 6.9 | 1.58 |
| 03:04 | 26.06 | 52 | 42 | 0.05 | 18 | 3 | 6.07 | 3.41 |
| 03:05 | 716.84 | 237 | 116 | 0.25 | 42 | 13 | 9.65 | 3.63 |
| 03:06 | 13256.21 | 803 | 157 | 0.36 | 62 | 21 | 3.87 | 1.57 |
| 04:05 | 25.34 | 50 | 40 | 0.05 | 19 | 4 | 6.38 | 2.51 |
| 04:06 | 357.72 | 174 | 73 | 0.19 | 32 | 6 | 6.1 | 2.89 |
| 04:07 | 4515.55 | 485 | 128 | 0.21 | 42 | 13 | 4.91 | 1.53 |
| 05:06 | 14.67 | 25 | 17 | 0.03 | 12 | 2 | 6.57 | 3.29 |
| 05:07 | 162.16 | 99 | 38 | 0.08 | 22 | 5 | 6.26 | 2.84 |
| 05:08 | 10447.3 | 810 | 179 | 0.34 | 58 | 20 | 5.3 | 1.96 |
| 06:07 | 8.11 | 16 | 12 | 0.03 | 9 | 2 | 6.27 | 3.13 |
| 06:08 | 314.52 | 141 | 71 | 0.12 | 30 | 4 | 4.74 | 3.15 |
| 06:09 | 3967.18 | 430 | 102 | 0.26 | 43 | 11 | 2.53 | 1.68 |
| 07:08 | 22.63 | 39 | 38 | 0.05 | 14 | 2 | $< 0.01$ | $< 0.01$ |
| 07:09 | 314.79 | 141 | 75 | 0.08 | 25 | 5 | 6.51 | 2.91 |
| 07:10 | 11293.63 | 703 | 211 | 0.27 | 41 | 12 | 3.1 | 1.44 |
| 08:09 | 23.27 | 38 | 29 | 0.04 | 16 | 3 | 7.53 | 4.49 |
| 08:10 | 625.71 | 247 | 125 | 0.14 | 33 | 9 | 6.47 | 2.57 |
| 08:11 | 10894.5 | 748 | 202 | 0.33 | 49 | 17 | 4.08 | 1.84 |
| 09:10 | 14.38 | 29 | 22 | 0.03 | 12 | 2 | 7.04 | 3.52 |
| 09:11 | 160.06 | 100 | 48 | 0.08 | 21 | 3 | 2.72 | 2.48 |
| 09:12 | 11093.29 | 728 | 211 | 0.29 | 48 | 18 | 3.7 | 1.68 |
| 10:11 | 14.31 | 29 | 23 | 0.04 | 12 | 2 | 8.9 | 4.45 |
| 10:12 | 623.01 | 248 | 141 | 0.18 | 36 | 12 | 6.0 | 2.55 |
| 10:13 | 10676.22 | 739 | 178 | 0.28 | 65 | 26 | 6.24 | 1.95 |
| 11:12 | 22.81 | 38 | 34 | 0.04 | 16 | 3 | 7.44 | 2.48 |
| 11:13 | 320.6 | 132 | 52 | 0.11 | 38 | 7 | 8.25 | 5.1 |
| 11:14 | 3344.99 | 449 | 84 | 0.24 | 53 | 18 | 8.77 | 3.52 |
| 12:13 | 22.59 | 35 | 24 | 0.04 | 20 | 1 | $< 0.01$ | $< 0.01$ |
| 12:14 | 329.11 | 129 | 47 | 0.11 | 37 | 7 | 7.38 | 4.08 |
| 13:14 | 8.21 | 14 | 8 | 0.03 | 9 | 0 | 0.00 | 0.00 |

Table 5.4.2: Comparison between exact and heuristic methods

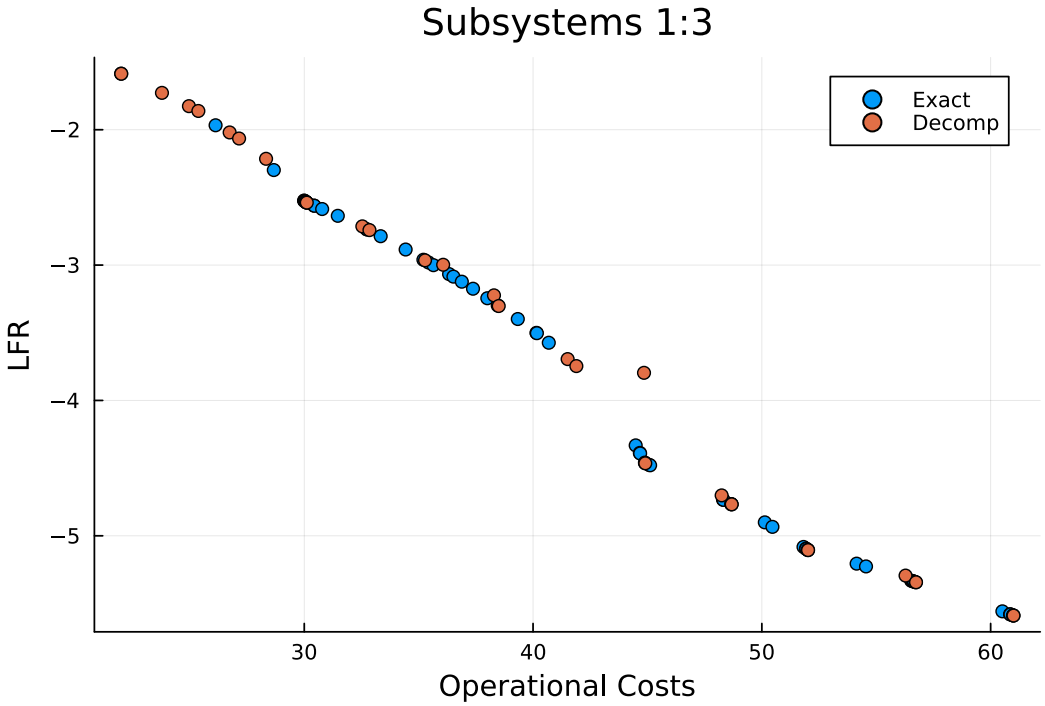



Figure 5.4.1: Exact and approximate Pareto fronts for problem 1:3

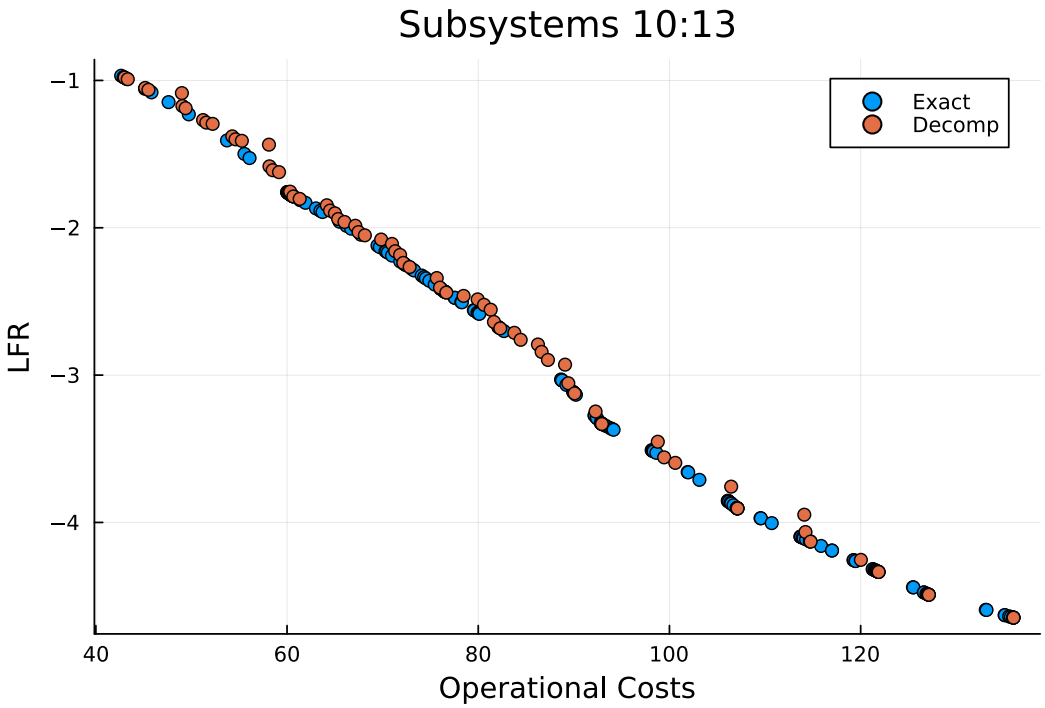



Figure 5.4.2: Exact and approximate Pareto fronts for problem 10:13

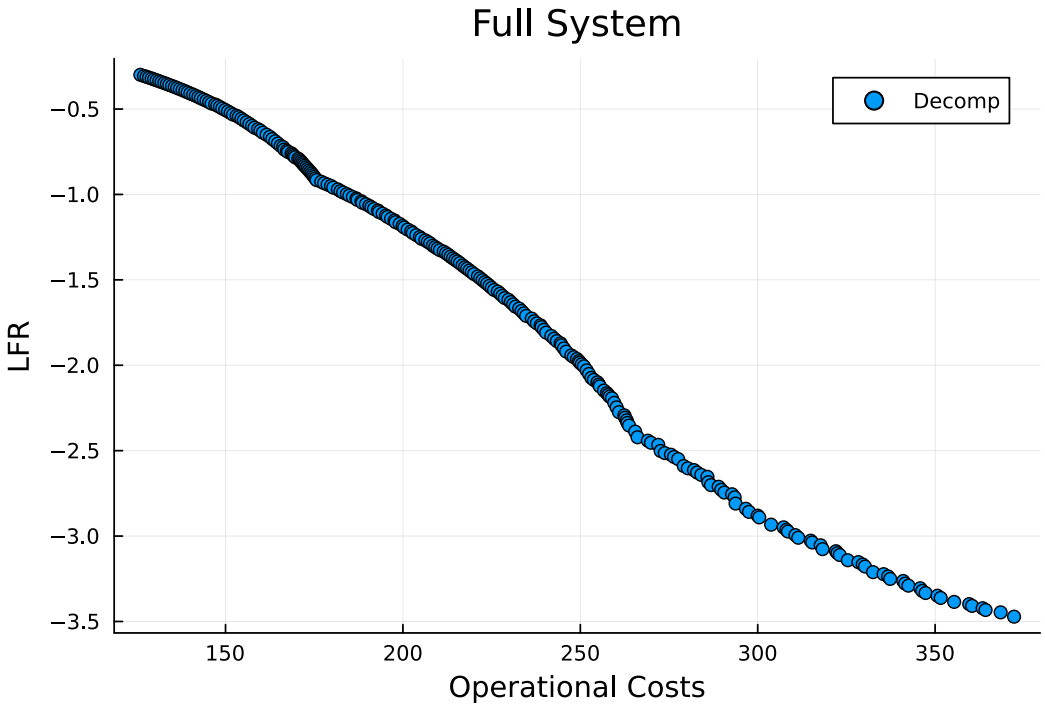



Figure 5.4.3: Approximate Pareto front for the full system

theory using equivalence relations to deal with the symmetries in these models. We have also identified an intuitive class of policies - the coherent policies - which always includes an optimal policy and possesses intuitive properties. These principles have also been extended to large impulsive CTMDPs, which can be regarded as being made up of smaller impulsive CTMDPs, and we have shown how to make use of our equivalence theory to analyse and heuristically optimise these models. We have also applied our modelling framework and heuristic solution methodology to a dynamic maintenance problem, and demonstrated that the heuristic performs well in terms of near-Pareto-optimality and scales much better than exact methods.

Currently, our framework lacks some of the flexibility of pre-existing frameworks of impulsive control for CTMDPs. Specifically, we do not allow for uncertainty in the outcomes of taking impulsive actions, we do not allow for lump-sum costs for taking impulsive actions, and we only allow for one gradual action (i.e., the zero action) per state. Further work could be done to add these features to our framework. However,

we argue that the simplicity of our framework, and its compatibility with out-of-the-box solution methods, makes it an attractive option for modellers. Additionally, the framework could be enhanced by considering modelling features not currently present in the impulsive control literature, such as constraints. Specifically, constraints that span the whole system could translate into linking constraints in the block-diagonal problem, necessitating an iterative column generation approach, even in the single objective case. Such a method could be applicable to weakly-coupled MDPs, for which it is common to relax instantaneous action constraints to long-run average or discounted constraints.

Further research could also consider other application areas in which it makes sense to model problems as impulsive or decomposable impulsive CTMDPs. For example, one could consider queueing control problems with many queues that can be treated independently, or inventory problems under continuous review with many products. Where our heuristic methodology is applicable, it could be tested and compared with exact optimal solutions in order to investigate the Pareto-optimality gap.

# Chapter 6

# Bi-Objective Integrated Design and Dynamic Maintenance Problem for Series-Parallel Systems

## 6.1 Introduction

In this chapter, we work towards merging the modelling and solution methodologies of Chapter 4 and Chapter 5 to model and optimise both the design and dynamic maintenance strategies of series-parallel systems. Namely, we build upon the Dantzig-Wolfe decomposition approach developed in Chapter 5 by extending it to the design case, inspired by previous uses of Dantzig-Wolfe decomposition in the RAP literature for both heuristic (Zia and Coit, 2010) and exact (Reihaneh et al., 2022) solution methodologies.

In Section 6.2 we provide the problem formulation as a MILP, extending the MDP Design problem of Chapter 4 to include multiple subsystems in series, along the same lines as the maintenance model in Section 5.4. In Section 6.3.1 we start by providing an efficient method for constructing the spaces of states, actions, and state-action pairs. In

Section 6.3.3 we then provide a range of solution methodologies for some different variations of the problem, starting with a simplified version of the problem in Section 6.3.4. This is used in turn as a starting point for a detailed iterative algorithm for the more general problem in Section 6.3.5, which considers a range of pricing subproblems. Section 6.4 provides some preliminary computational results. We then demonstrate how quickly the model becomes impossible to solve exactly, even with our efficient model construction, and then compare the exact solver against a partial implementation of our heuristic for small, solvable problem instances, yielding encouraging results. We also demonstrate that the heuristic scales well to larger problem instances which would be impossible to solve exactly.

### 6.1.1 Our Contributions

The contributions of this chapter are as follows:

- In Section 6.2 we provide the first optimisation model that integrates both design and dynamic maintenance decisions into a unified model for series-parallel systems.
- In Section 6.3 we provide an algorithm for the efficient construction of the full state, action, and state-action spaces for an MDP Design problem with constraints of the type considered. This is required for the exact solver.
- In Section 6.3 we also demonstrate the first application of Dantzig-Wolfe decomposition to an MDP Design problem with a block-diagonal structure, and leverage this to design some heuristic and near-exact algorithms for our model.
- In Section 6.4 we explore the rapid rate at which solution times for the exact solver for our problem grow with respect to the number of subsystems, and demonstrate a partial implementation of a decomposition-based heuristic.

## 6.2 Problem Formulation

In this section, we build towards a precise definition of our model. In Section 6.2.1, we give a brief overview of the decomposable impulsive CTMDP framework. In Section 6.2.2, we define our dynamic maintenance problem for a series-parallel system with heterogeneous components, first doing this for a parallel system, and then extending this to the series-parallel case. In Section 6.2.3, we extend this further into an MDP Design problem.

### 6.2.1 Decomposable Impulsive CTMDPs

We apply the decomposable impulsive CTMDP framework of Chapter 5 for our dynamic maintenance problem, and first recall some of the definitions. A *bi-objective CTMDP with self-transitions* is defined as a tuple:

$$\mathfrak{C} = \langle \mathcal{S}, (\mathcal{A}(s))_{s\in\mathcal{S}}, q^+ : \mathcal{SA} \times \mathcal{S} \to \mathbb{R}_{\geq 0}, \boldsymbol{c} : \mathcal{SA} \to \mathbb{R}^2 \rangle,$$

where $\mathcal{S}$ is a *state space*, and for each state $s \in \mathcal{S}$ we have an *action space* $\mathcal{A}(s)$. We can then define the feasible state-action space as $\mathcal{SA} = \{(s,a) : s \in \mathcal{S}, a \in \mathcal{A}(s)\}$. The function $q^+$ gives the *generalised transition rates* and the vector function $\boldsymbol{c}$ gives the vector cost rates. We can then define *negative generalised transition rates* as $q^-(s,a,s') = -\mathbb{I}\{s = s'\} \sum_{s''\in\mathcal{S}} q^+(s,a,s'')$ to give either the total outbound transition rate from the state-action pair for $s = s'$, and zero otherwise. Then, we recover the standard transition rate function $q$ from $q = q^+ + q^-$. The reason we use this alternative modelling framework is that it's easier when dealing with *impulsivity*, where states can take on the transition rates of other states, which can lead to confusion when it comes to outbound and inbound rates. To satisfy impulsivity, there must exist a "zero" action denoted by 0 in every feasible action space $\mathcal{A}(s)$, and an operator $\oplus : \mathcal{SA} \to \mathcal{S}$ satisfying the following:

- $s \oplus 0 = s$ for all $s \in \mathcal{S}$,
- If $a_1 \in \mathcal{A}(s), a_2 \in \mathcal{A}(s \oplus a_1), a_1, a_2 \neq 0$, then there exists $a_1 \cdot a_2 \in \mathcal{A}(s)$ such that $s \oplus (a_1 \cdot a_2) = (s \oplus a_1) \oplus a_2$,
- $q^+(s, a, s') = q^+(s \oplus a, 0, s')$ for all $s, a, s'$,
- $\boldsymbol{c}(s, a) = \boldsymbol{c}(s \oplus a, 0)$, for all $s, a$.

We can then define $q^+(s, s') := q^+(s, 0, s')$ and $\boldsymbol{c}(s) := \boldsymbol{c}(s, 0)$ to avoid redundancy. The tuple $\mathfrak{C} = \langle \mathcal{S}, (\mathcal{A}(s))_{s\in\mathcal{S}}, q^+ : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}, \boldsymbol{c} : \mathcal{S} \to \mathbb{R}^2, \oplus : \mathcal{SA} \to \mathcal{S} \rangle$ is called an *impulsive* CTMDP.

We now want a way of constructing an impulsive CTMDP which is made up of multiple impulsive CTMDPs (in our case, we will want to define a series-parallel maintenance problem by composing together multiple parallel maintenance problems). Given an indexed set of costless impulsive CTMDPs $\{\mathfrak{C}_i\}_{i=1}^I$ with $\mathfrak{C}_i = \langle \mathcal{S}_i, (\mathcal{A}_i(s_i))_{s_i\in\mathcal{S}_i}, q_i^+ : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}, \oplus_i : \mathcal{SA}_i \to \mathcal{S}_i \rangle$, the *cost-free direct product* of these impulsive CTMDPs is the costless impulsive CTMDP:

$$\bigotimes_{i=1}^{I} \mathfrak{C}_i := \langle \mathcal{S}, (\mathcal{A}(s))_{s\in\mathcal{S}}, q^+ : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}, \oplus : \mathcal{SA} \to \mathcal{S} \rangle,$$

satisfying:

$$\begin{aligned}
\mathcal{S} &= \bigtimes_{i=1}^{I} \mathcal{S}_i, \\
\mathcal{A}(s) &= \bigtimes_{i=1}^{I} \mathcal{A}_i([s]_i), \\
q^+(s, s') &= \begin{cases} q_i^+([s]_i, s_i'), & s' = ([s]_1, ..., s_i', ..., [s]_I) \text{ for some } i, \\ 0, & \text{otherwise,} \end{cases} \\
s \oplus a &= ([s]_1 \oplus_1 [a_1], ..., [s]_I \oplus_I [a]_I).
\end{aligned}$$

Additionally, we refer to $\mathfrak{C}_i$ as the $i^{th}$ *compartment* of the product CTMDP. The definition of $q$ ensures that after composing the multiple CTMDPs together, an update can only ever happen in one of the compartments of the state, in a way that respects the independence of the original processes and the original transition rates. Section 6.2.2 now follows by applying this framework to define the series-parallel maintenance problem.

### 6.2.2 Dynamic Maintenance Problem

We now provide the formulation of the series-parallel maintenance problem (SPMP) for a fixed design. For a system with $I$ subsystems, we will do this by taking the direct product of impulsive CTMDPs $\{\mathfrak{C}_i\}_{i=1}^{I}$ for the $I$ subsystems as though they were separate. We want to include impulsivity because, while the repairs themselves are not instantaneous, the decision to make a repair (e.g. phoning a repair worker) is, and this decision locks in the fact that this repair will happen. The time to complete a repair can then be interpreted as the total time between making the phone call and the component becoming operational.

Each subsystem CTMDP $\mathfrak{C}_i$ has $J_i$ different component types, and $K_{i,j}$ copies of component type $j$. The state space for a given subsystem is $\mathcal{S}_i = \{\boldsymbol{v} \in \mathbb{Z}_{\geq 0}^{J_i \times 2} : v_{j,1} + v_{j,2} \leq K_{i,j}\}$, so a state $\boldsymbol{v}$ of the subsystem is a component-wise non-negative $J_i \times 2$ integer matrix, where the entry $v_{j,1}$ describes the number of ongoing repairs on components of type $j$, and $v_{j,2}$ gives the number of damaged components of type $j$ that are not currently being repaired. For notational convenience, we define $v_{j,0} := K_{i,j} - v_{j,1} - v_{j,2}$ as the number of healthy components for a given type. The feasible action spaces are then given by $\mathcal{A}_i(\boldsymbol{v}) := \{\boldsymbol{b} \in \mathbb{Z}_{\geq 0}^{J_i} : b_j \leq v_{j,2}\}$, so an action $\boldsymbol{b}$ is a component-wise non-negative integer vector of length $J_i$ whose $j^{\text{th}}$ entry describes the number of repairs to be started on components of type $j$, and this action is clearly limited by the number of damaged components of this type, $v_{j,2}$. Our impulsive operator $\oplus_i$ is then defined as $\boldsymbol{v} \oplus_i \boldsymbol{b} := v + B(\boldsymbol{b})$, where $B(\boldsymbol{b})$ is a matrix of equal dimension to $\boldsymbol{v}$ such that

$[B(\boldsymbol{b})]_{j,1} = b_j$, $[B(\boldsymbol{b})]_{j,2} = -b_j$, and all other entires are zero. Intuitively, if we decide to repair a non-zero number $b_j$ of component type $j$, we take $b_j$ components out of the damaged condition $v_{j,2}$, and move them into the repairing condition $v_{j,1}$. It is clear that our "zero" action is the vector of zeroes of length $J_i$, and that if $\boldsymbol{b}_1 \in \mathcal{A}_i(\boldsymbol{v})$, $\boldsymbol{b}_2 \in \mathcal{A}_i(\boldsymbol{v} \oplus \boldsymbol{b}_1)$, $\boldsymbol{b}_1, \boldsymbol{b}_2 \neq \boldsymbol{0}$, then there exists $\boldsymbol{b}_1 \cdot \boldsymbol{b}_2 = \boldsymbol{b}_1 + \boldsymbol{b}_2 \in \mathcal{A}_i(\boldsymbol{v})$ such that $\boldsymbol{v} \oplus (\boldsymbol{b}_1 + \boldsymbol{b}_2) = (\boldsymbol{v} \oplus \boldsymbol{b}_1) \oplus \boldsymbol{b}_2$, so these necessary characteristics of the impulsive operator are satisfied.

It remains now to define the stochastic dynamics and costs of the subsystems. We need only define the functions $q_i^+ : \mathcal{S}_i \times \mathcal{S}_i \to \mathbb{R}_{\geq 0}$ and $\boldsymbol{c}_i : \mathcal{S}_i \to \mathbb{R}^2$ which are the generalised transition rates and bi-objective costs for the zero action, as impulsivity handles the other actions. To aid in the definition of $q_i^+$, we will introduce matrices $\boldsymbol{e}_{j,1}$ and $\boldsymbol{e}_{j,2}$ which are matrices of dimension equal to $v$ whose entries are zero everywhere except entry $(j,1)$ or $(j,2)$, where it is 1. We assume that there are two types of event: component degradation, and repair completion. For a component of type $j$, the time until failure is assumed to be exponentially distributed with rate $\alpha_{i,j}$, and the time until repair completion is assumed to be exponentially distributed with rate $\tau_{i,j}$.

Following the above, we define $q_i^+$ as follows:

$$q_i^+(\boldsymbol{v}, \boldsymbol{v}') = \begin{cases} v_{j,0}\alpha_{i,j}, & \boldsymbol{v}' = \boldsymbol{v} + \boldsymbol{e}_{j,2}, \\ v_{j,1}\tau_{i,j}, & \boldsymbol{v}' = \boldsymbol{v} - \boldsymbol{e}_{j,1}, \\ 0, & \text{otherwise.} \end{cases}$$

The first line describes a transition where a healthy component becomes damaged, and the second line describes a transition where a repairing component completes that repair and becomes healthy.

Finally, we describe the subsystem cost functions. There are three sources of cost: repair costs $r_{i,j}$, incurred per unit time when performing maintenance on a component;

usage costs $u_{i,j}$, incurred per unit time when a given component is the cheapest available healthy component; and subsystem failure, which is an indicator function that tells us if a subsystem has failed. The first two pertain to the first objective (monetary cost), and the third pertains only to the second objective (reliability). Without loss of generality, for any subsystem $i$, we assume that the $J_i$ components are ordered such that the usage costs $u_{i,j}$ are in ascending order, i.e., for every $i$ we have $u_{i,1} \leq u_{i,2} \leq ... \leq u_{i,J_i}$. We then define our operational costs as follows:

$$c_i^o(\boldsymbol{v}) := \sum_{j=1}^{J_i} r_{i,j} v_{j,1} + \min_{j:v_{j,0}>0} u_{i,j}.$$

This means that the operational cost for any given state is the sum of all the ongoing repair costs for each copy of each component currently being repaired, plus the usage cost of the cheapest available component. We use the convention that $\min \emptyset = 0$ for states with no healthy components. Our failure indicator cost is then defined as follows:

$$c_i^f(\boldsymbol{v}) = \mathbb{I}\left\{\sum_{j=1}^{J_i} v_{j,0} = 0\right\}.$$

Simply, the system has failed if the total number of healthy components of any type is 0.

We now need to combine these subsystem CTMDPs into one model, and define the cost structure. We obtain the state space $\mathcal{S}$, action spaces $\mathcal{A}(s)$, transition rates $q^+$, and impulsive operator $\oplus$ by taking the cost-free direct product of the impulsive CTMDPs $\{\mathfrak{C}_i\}_{i=1}^I$. As such, the separate subsystems remain stochastically independent (under the zero action), but are represented together as one unified CTMDP. All that remains is to define the cost structure $\boldsymbol{c} = (c^o, c^f)$ of the product CTMDP. The operational

costs are simply treated additively, so we have:

$$c^o(\boldsymbol{s}) = \sum_{i=1}^{I} c_i^o([\boldsymbol{s}]_i),$$

for all $\boldsymbol{s} \in \mathcal{S}$, where this works under the simplifying assumption that usage costs continue to be incurred from subsystems with healthy components, even if there is a subsystem that is non-operational. This may seem unrealistic, as a more complex model may have the other subsystems shut down and not incur usage costs when the system has failed overall. Despite this, for a reliable system that is very often operational, we would expect the long-run usage costs in either model to be similar to the other, as any sort of failure state should be quite rare. Our other type of cost rate is the failure indicator cost, which should indicate when *at least one* of the subsystems has failed, which we express as a *logical-OR* cost function as follows:

$$c^f(\boldsymbol{s}) = \mathbb{I}\left\{\bigvee_{i=1}^{I} c_i^f([\boldsymbol{s}]_i) = 1\right\}.$$

The problem of finding all Pareto-optimal randomised stationary policies can be represented by the following bi-objective LP, which we call BO-SPMP:

$$\begin{aligned}
\text{(BO-SPMP)} \quad \min_{\boldsymbol{\pi}} & \sum_{(s,a)\in\mathcal{SA}} c^o(\boldsymbol{s},\boldsymbol{a})\pi(\boldsymbol{s},\boldsymbol{a}) && (6.2.1)\\
\min_{\boldsymbol{\pi}} & \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}} c^f(\boldsymbol{s},\boldsymbol{a})\pi(\boldsymbol{s},\boldsymbol{a}) && (6.2.2)\\
\text{s.t.} & \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}} q(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}')\pi(\boldsymbol{s},\boldsymbol{a}) = 0 \text{ for all } \boldsymbol{s}' \in \mathcal{S} && (6.2.3)\\
& \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}} \pi(\boldsymbol{s},\boldsymbol{a}) = 1 && (6.2.4)\\
& \pi(\boldsymbol{s},\boldsymbol{a}) \geq 0, \text{for all } (\boldsymbol{s},\boldsymbol{a}) \in \mathcal{SA}. && (6.2.5)
\end{aligned}$$

The decision variables $\boldsymbol{\pi}$ are the state-action frequencies, i.e., the long-run average proportion of time that the process is in a given state and taking a certain action. Objectives (6.2.1) and (6.2.2) give the cost and failure objectives, constraint (6.2.3) gives the CTMDP balance equations, and constraints (6.2.4) and (6.2.5) ensure $\boldsymbol{\pi}$ is a valid probability distribution. Any feasible solution $\boldsymbol{\pi}$ corresponds to a randomised policy $\mu(\boldsymbol{s}, \boldsymbol{a}) = \frac{\pi(\boldsymbol{s},\boldsymbol{a})}{\sum_{\boldsymbol{a}' \in \mathcal{A}(\boldsymbol{s})} \pi(\boldsymbol{s},\boldsymbol{a}')}$ where $\mu(\boldsymbol{s}, \boldsymbol{a})$ is the probability of taking action $a$ when in state $s$. Any basic feasible solution $\boldsymbol{\pi}$ corresponds to a deterministic stationary policy $\mu(\boldsymbol{s}) = \arg\max_{\boldsymbol{a} \in \mathcal{A}(\boldsymbol{s})} \pi(\boldsymbol{s}, \boldsymbol{a})$. As we have done in previous chapters, we focus on finding Pareto-optimal basic feasible solutions of BO-SPMP, i.e., DS policies that are Pareto-optimal amongst all randomised stationary policies. We now follow with an MDP Design extension of this problem formulation, which includes binary design variables.

### 6.2.3 Integrated Design and Dynamic Maintenance Problem

We want to formulate a problem in which we first choose how many copies of each component to be installed, subject to $L$ knapsack constraints $\sum_{i=1}^{I} \sum_{j=1}^{J_i} C_{i,j,l} y_{i,j} \leq d_l$ for $l = 1, ..., L$, where $C_{i,j,l} \geq 0$ is the contribution of one unit of component type $j$ for subsystem $i$ to constraint $l$, $y_{i,j}$ is the integer number of copies of component type $j$ in subsystem $i$, and $d_l$ is the limit for constraint $l$. Define $\boldsymbol{y}_i = (y_{i,1}, ..., y_{i,J_i})$ and obtain a vectorisation of the $\boldsymbol{y}$ variables as $y = (y_1, ..., y_I)$. We then write the knapsack contributions in matrix form as $[C]_{l,(i,j)} = C_{i,j,l}$, and write the constraints simply as $C\boldsymbol{y} \leq \boldsymbol{d}$. Simultaneously, in the same optimisation problem, we also wish to dynamically maintain the resulting design. We call this the *Series-Parallel Integrated Design and Maintenance Problem*, or SPIDMP. To formalise this problem, we first formulate a CTMDP that has the maximum number of copies of each component, $K_{i,j}^{\max}$. Finding this is an optimisation problem itself, as for a problem with $I$ subsystems each

with $J_i$ component types we have:

$$K_{i,j}^{\max} = \max_{\boldsymbol{y}} \left\{ y_{i,j} : C\boldsymbol{y} \leq \boldsymbol{d}, \sum_{j=1}^{J_i} y_{i',j} = 1 \text{ for all } i' \neq i, i' \in \{1, ..., I\}, y_{i,j} \in \mathbb{Z}_{\geq 0} \text{ for all } i, j \right\}.$$

As such, $K_{i,j}^{\max}$ is defined to be the maximum number of copies of component type $j$ in subsystem $i$ such that there is exactly one component in each other subsystem, and this set of components satisfies the knapsack constraints. We can then use this to define a CTMDP with the maximum number of copies of each component. This is done via the construction found in Section 6.2.2, by first defining subsystem CTMDPs with maximal state spaces as $\mathcal{S}_i^{\max} = \{\boldsymbol{v} \in \mathbb{Z}_{\geq 0}^{J_i \times 2} : v_{j,1} + v_{j,2} \leq K_{i,j}^{\max}\}$, whose action spaces, transition rates, costs, and impulsive operators follow the previous definition. The series-parallel problem is then obtained via the direct product as before, notably yielding the full-system state space $\mathcal{S}^{\max} = \bigtimes_{i=1}^{I} \mathcal{S}_i^{\max}$ and feasible state-action space $\mathcal{SA}^{\max} = \{(\boldsymbol{s}, \boldsymbol{a}) : \boldsymbol{s} \in \mathcal{S}^{\max}, \boldsymbol{a} \in \mathcal{A}(\boldsymbol{s})\}$. With our maximal CTMDP defined, we can modify its LP formulation to include binary decision variables and suitable constraints that ensure that only state-action frequencies for states and actions that conform with our design choices may take a non-zero value. To do this cleanly, we switch from general integer variables $\boldsymbol{y}$ to binary variables $\boldsymbol{x}$ where $x_{i,j,k}$ is the decision to install the $k^{\text{th}}$ copy of component type $j$ in subsystem $i$. We also define a new binary-compatible constraint matrix $C'$ such that $[C']_{l,(i,j,k)} = [C]_{l,(i,j)}$. We provide the following problem

formulation:

$$\text{(BO-SPIDMP)} \quad \min_{\boldsymbol{\pi},\boldsymbol{x}} \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}^{\max}} c^o(\boldsymbol{s},\boldsymbol{a})\pi(\boldsymbol{s},\boldsymbol{a}) \tag{6.2.6}$$

$$\min_{\boldsymbol{\pi},\boldsymbol{x}} \sum_{(s,a)\in\mathcal{SA}^{\max}} c^f(\boldsymbol{s},\boldsymbol{a})\pi(\boldsymbol{s},\boldsymbol{a}) \tag{6.2.7}$$

$$\text{s.t. } (6.2.3) - (6.2.5), \tag{6.2.8}$$

$$C'\boldsymbol{x} \leq \boldsymbol{d}, \tag{6.2.9}$$

$$\sum_{\substack{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}^{\max}:\\ [\boldsymbol{s}\oplus\boldsymbol{a}]_{i,j,2}\leq K^{\max}_{i,j}-k}} \pi(\boldsymbol{s},\boldsymbol{a}) \leq x_{i,j,k}, \text{ for all } i,j,k, \tag{6.2.10}$$

$$x_{i,j,k} - x_{i,j,k+1} \geq 0, \text{ for all } i,j, \text{ and } k \leq K^{\max}_{i,j} - 1, \tag{6.2.11}$$

$$\sum_{j=1}^{J_i}\sum_{k=1}^{K^{\max}_{i,j}} x_{i,j,k} \geq 1, \tag{6.2.12}$$

$$x_{i,j,k} \in \{0,1\} \text{ for all } i,j,k. \tag{6.2.13}$$

As with BO-SPMP, the decision variables include the state-action frequencies $\pi$, and we now include the binary design variables $x$. Objectives (6.2.6) and (6.2.7) are again the cost and failure objectives, respectively, then constraints (6.2.8) simply bring over the CTMDP constraints of BO-SPMP for the maximal CTMDP, and constraint (6.2.9) includes the knapsack constraints. Constraint (6.2.10) means that we can only spend a non-zero long-run proportion of time in a state-action pair whose target state $\boldsymbol{s}\oplus\boldsymbol{a}$ has less than $K^{\max}_{i,j}-k$ damaged copies of component type $j$ in subsystem $i$ (or equivalently, more than $k$ copies healthy or repairing) if the $k^{\text{th}}$ copy of that component (and, by the symmetry-breaking constraint (6.2.11), the previous $k-1$ copies) have all been included in the design. This works by using the fact that having a damaged component that you can't repair is the same as not having that component at all. Constraint (6.2.12) ensures that we have at least one component per subsystem. Finally, constraint (6.2.13) ensures all $\boldsymbol{x}$ variables are binary.

It is clear that the maximal state space grows exponentially both in the number of subsystems and in the number of component types, as do the action spaces. As such, the problem rapidly grows in size even for examples with a relatively small number of subsystems and component types. However, we may identify many states, actions, and state-action pairs to be excluded from the problem, as they are infeasible for any possible design. We define the *minimal design* for a state $\boldsymbol{s}$ (in general integer variables) as a structure $\boldsymbol{y}(\boldsymbol{s})$ with indices $(i,j)$ for $i \in [I], j \in [J_i]$ such that $[\boldsymbol{y}(\boldsymbol{s})]_{i,j} = K^{\max}_{i,j} - s_{i,j,2}$. We also define the feasible design space:

$$\mathcal{Y} = \left\{ \boldsymbol{y} \in \bigtimes_{i=1}^{I} \mathbb{Z}^{J_i}_{\geq 0} : C\boldsymbol{y} \leq \boldsymbol{d}, \sum_{j=1}^{J_i} y_{i,j} \geq 1 \text{ for all } i. \right\}.$$

Note that if a state $s$ has a failed subsystem $i$ with no ongoing repairs, then $[\boldsymbol{y}(\boldsymbol{s})]_i = \boldsymbol{0}$, and in this case $\boldsymbol{y}(\boldsymbol{s}) \notin \mathcal{Y}$. As such, if we want to check that such an $\boldsymbol{s}$ is feasible, we must check for the existence of a $\boldsymbol{y}' \in \mathcal{Y}$ such that $\boldsymbol{y}' \geq \boldsymbol{y}(\boldsymbol{s})$, where $\geq$ is taken component-wise. We can then define the following knapsack-constrained state, action, and state-action spaces:

$$\begin{aligned}
\mathcal{S}^{\max}_{\text{knap}} &:= \{\boldsymbol{s} \in \mathcal{S}^{\max} : \exists \boldsymbol{y}' \in \mathcal{Y} \text{ such that } \boldsymbol{y}' \geq \boldsymbol{y}(\boldsymbol{s})\}, \\
\mathcal{A}_{\text{knap}}(\boldsymbol{s}) &:= \{\boldsymbol{a} \in \mathcal{A}(\boldsymbol{s}) : \boldsymbol{s} \oplus \boldsymbol{a} \in \mathcal{S}^{\max}_{\text{knap}}\}, \\
\mathcal{SA}^{\max}_{\text{knap}} &:= \{(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{SA}^{\max} : \boldsymbol{s} \oplus \boldsymbol{a} \in \mathcal{S}^{\max}_{\text{knap}}\}.
\end{aligned}$$

As $\pi(\boldsymbol{s}, \boldsymbol{a}) = 0$ for all $(\boldsymbol{s}, \boldsymbol{a})$ not in $\mathcal{SA}^{\max}_{\text{knap}}$ for any feasible design, we may substitute $\mathcal{S}^{\max}_{\text{knap}}, \mathcal{A}_{\text{knap}}(s), \mathcal{SA}^{\max}_{\text{knap}}$ for $\mathcal{S}^{\max}, \mathcal{A}(s), \mathcal{SA}^{\max}$ respectively in BO-SPIDMP, yielding a problem that may have significantly fewer variables. Whilst this allows us to more easily obtain some exact results for BO-SPIDMPs, the problem still blows up in scale in the number of component types and number of subsystems. Additionally, the MDP Design problem is not well suited to any standard metaheuristic methodologies for

large MILPs, as the difficulty stems from the exponentially large number of continuous variables rather than the integer variables, and we therefore require a bespoke solution methodology. Section 6.3 now follows with an investigation of exact and heuristic solution methodologies for this problem.

## 6.3 Methodology

### 6.3.1 Near-Exact

The first choice to be made when deciding on how to solve our bi-objective problem exactly is how to scalarise it, so we must choose between the mixed objectives method and the $\varepsilon$-constraint method. While the former suffers from missing non-supported solutions, the latter interferes with the structure of the MDP, allowing it to return solutions with randomised policies, which we consider to be undesirable. As such, we proceed by using the mixed objectives method, as we do not know of an efficient method that both finds all supported and non-supported solutions and is limited to deterministic policies. Specifically, we use an adjusted version of the dichotomic search method of Aneja and Nair (1979), which is a method for automatically calculating the objective weightings in such a way that, for all Pareto-optimal supported solutions that are optimal for some weighting, such a weighting will be evaluated by the method. Note that this is not the same as finding all Pareto-optimal supported solutions: if multiple solutions are optimal only for one given weighting, then we are only guaranteed to find one of those solutions for that weighting when solving using an LP solver. As such, we only call our method "near-exact", as the solutions found are Pareto-optimal, but not all such solutions are guaranteed to be found. Similarly to Chapter 4, we use a slightly adjusted version of the dichotomic search with two changes: first, weights are normalised so that the cost objective always gets a weight of 1, and a large penalty is given to the failure objective; and second, nodes with numerical issues are fathomed.

To elaborate on the latter, we have encountered issues regarding nodes $(w_1, w_2)$ where the solutions for $w_1$ and $w_2$ are extremely numerically similar in terms of their objective values. When calculating the new weight for such a node, we have encountered "not a number" errors, or found that the new weight has not laid between $w_1$ and $w_2$ due to numerical error. The former leads to a crash, and the latter leads to cycling. As such, when this happens, we simply fathom the node, which we consider to be reasonable as this only happens when the solutions are very similar in the first place, so very little is lost in terms of the completeness of the Pareto front. Each scalarisation is of course solved using a MILP solver to get an optimal solution for a given node.

The other challenge of making an exact solver is the construction of $\mathcal{S}^{\max}_{\text{knap}}, \mathcal{A}_{\text{knap}}(s)$, and $\mathcal{SA}^{\max}_{\text{knap}}$; namely, we want to accomplish this efficiently, without ever explicitly building $\mathcal{S}^{\max}$. Recall that we encountered a similar problem in Section 4.2.2, and solved this via a dynamic programming (i.e., bottom-up) approach in Section A.1.2. We work along similar lines here, but must account for the multiple subsystems and the knapsack constraints that link them. Firstly, the one-subsystem one-component-type partial state spaces for each subsystem $i$ and each component type $j$ are defined as $\mathcal{S}^{\max}_{i,j} = \left\{(s_{i,j,1}, s_{i,j,2}) \in \mathbb{Z}^2_{\geq 0} : s_{i,j,1} + s_{i,j,2} \leq K^{\max}_{i,j}\right\}$, and can easily be constructed due to their low dimensionality. Next, we wish to consider the partial state spaces $\mathcal{S}^{\max}_{i,1:j}$, which we informally describe as the set of partial states for subsystem $i$ that use the first $j \leq J_i$ component types, and which leave enough knapsack capacity for the rest of the system to have at least one component part subsystem. Let $\boldsymbol{s}_{i,1:j} \in \bigtimes_{j'=1}^{j} \mathcal{S}^{\max}_{i,j'}$ be a partial state for the $i^{\text{th}}$ subsystem with component types $1:j$, then we define $\boldsymbol{y}_{i,1:j}(s_{i,1:j}) = \left(K^{\max}_{i,j} - [\boldsymbol{s}_{i,1:j}]_{j',2}\right)^{j}_{j'=1}$ to be the *minimal partial design* (in general integer form, not binary) for the partial state, which is a $j$-dimensional vector whose components are the number of copies of a component type that must be installed for the state to be allowable, i.e., the number of components either healthy or repairing in the partial state $\boldsymbol{s}_{i,1:j}$. Using the feasible decision space $\mathcal{Y}$ as previously defined, we can

then define:

$$\mathcal{S}_{i,1:j}^{\max} := \left\{ \boldsymbol{s}_{i,1:j} \in \bigtimes_{j'=1}^{j} \mathcal{S}_{i,j}^{\max} : \exists \boldsymbol{y'} \in \mathcal{Y} \text{ such that } [\boldsymbol{y'}]_{i,1:j} \geq \boldsymbol{y}_{i,1:j}(\boldsymbol{s}_{i,1:j}) \right\}.$$

This is useful because we can notice the recursion:

$$\mathcal{S}_{i,1:j}^{\max} := \left\{ \boldsymbol{s}_{i,1:j} \in \mathcal{S}_{i:1:j-1}^{\max} \times \mathcal{S}_{i,j}^{\max} : \exists \boldsymbol{y'} \in \mathcal{Y} \text{ such that } [\boldsymbol{y'}]_{i,1:j} \geq \boldsymbol{y}_{i,1:j}(\boldsymbol{s}_{i,1:j}) \right\}.$$

This would allow us to sequentially construct $\mathcal{S}_{i,1:1}^{\max}$, $\mathcal{S}_{i,1:2}^{\max}$ up to $\mathcal{S}_{i,1:J_i}^{\max}$, where this last set would be all partial states $\boldsymbol{s}_i$ for subsystem $i$ for all component types for which there exists a whole-system design that permits this state (i.e., for which there exists a feasible solution to BO-SPIDMP with a whole-system state-action pair $(\boldsymbol{s}, \boldsymbol{a})$ with $[\boldsymbol{s}]_i = \boldsymbol{s}_i$ such that $\pi(\boldsymbol{s}, \boldsymbol{a}) > 0$). However, in practice, checking the existence of a $\boldsymbol{y} \in \mathcal{Y}$ when computing these partial state spaces requires the solving of a combinatorial feasibility problem for every potential partial state (i.e., every element of $\mathcal{S}_{i,1:j-1}^{\max} \times \mathcal{S}_{i,j}^{\max}$), and this would be far too computationally intensive. Instead, we recursively define some more easily computed supersets of the $\mathcal{S}_{i,1:j}^{\max}$:

$$\mathcal{S}_{i,1:1}^{\max'} := \mathcal{S}_{i,1}^{\max},$$
$$\mathcal{S}_{i,1:j}^{\max'} := \left\{ \boldsymbol{s}_{i,1:j} \in \mathcal{S}_{i,1:j-1}^{\max'} \times \mathcal{S}_{i,j}^{\max} : C_{i,1:j} \boldsymbol{y}_{i,1:j}(\boldsymbol{s}_{i,1:j}) \leq \boldsymbol{d} - \left( \sum_{i' \neq i} \min_{j'} C_{i',j',l} \right)_{l=1}^{L} \right\}.$$

So, when constructing the sets for subsystem $i$ sequentially, we use a relaxation that, in terms of the constraints, all other subsystems $i$ can simultaneously use their cheapest component for each constraint $l$. This of course leads to larger sets, but is far cheaper to compute, and as such serves as a reasonable middle ground. To construct each set, $\mathcal{S}_{i,j-1}^{\max'} \times \mathcal{S}_{i,j}^{\max}$ has each of its elements explicitly constructed and then compared against the constraint.

We can construct $\mathcal{S}_{i,1:J_i}^{\max'}$ for each $i$ separately using this method, and now call these

sets $\mathcal{S}^{\max'}_{i,\text{knap}}$. We now want to stick these sets together to obtain the feasible whole-system states, plus some infeasible states due to our approximation. First, consider the exact construction case. Let $\mathcal{S}^{\max}_{1:1} := \mathcal{S}^{\max}_{1,\text{knap}}$, then recursively define the *$i^{th}$-subsystem partial state spaces* as:

$$\mathcal{S}^{\max}_{1:i} = \left\{ \boldsymbol{s}_{1:i} \in \mathcal{S}^{\max}_{1:i-1} \times \mathcal{S}^{\max}_{i,\text{knap}} : \exists \boldsymbol{y'} \in \mathcal{Y} \text{ such that } [\boldsymbol{y'}]_{1:i} \geq \boldsymbol{y}_{1:i}(s_{1:i}) \right\}.$$

This recursion then satisfies $\mathcal{S}^{\max}_{1:I} = \mathcal{S}^{\max}_{\text{knap}}$, which is our desired set. Of course, in practice we do not assume access to the $\mathcal{S}^{\max}_{i,\text{knap}}$, and we want to avoid checking the existence condition. Instead, we construct the following supersets:

$$\mathcal{S}^{\max'}_{1:1} := \mathcal{S}^{\max'}_{1,\text{knap}},$$
$$\mathcal{S}^{\max'}_{1:i} := \left\{ \boldsymbol{s}_{1:i} \in \mathcal{S}^{\max'}_{1:i-1} \times \mathcal{S}^{\max'}_{i,\text{knap}} : C_{1:i}\boldsymbol{y}_{1:i}(\boldsymbol{s}_{1:i}) \leq \boldsymbol{d} - \left( \sum_{i'>i} \min_j C_{i',j,l} \right)_{l=1}^{L} \right\}.$$

Following this construction, $\mathcal{S}^{\max'}_{1:I}$, which we call $\mathcal{S}^{\max'}_{\text{knap}}$, gives a superset of $\mathcal{S}^{\max}_{\text{knap}}$ which is still far smaller than $\mathcal{S}^{\max}$ and comparatively easy to compute. Every state in $\mathcal{S}^{\max'}_{\text{knap}}$ will be knapsack-feasible by construction; however, it may not be feasible in the sense of allowing for one component per subsystem. For example, we may have a state where all components in a subsystem are damaged, and that subsystem therefore has no contributions to the knapsack constraints; however, repairing any one component in that subsystem would result in a knapsack-violating state. That is, $\mathcal{S}^{\max'}_{\text{knap}}$ will include states where the healthy and repairing components in the non-failed subsystems use of all of one of the knapsack constraints, and making repairs in one of the failed subsystems would take us over that knapsack limit. To put a name to this and be precise, we say a state $s$ is *quasi-feasible* if $C\boldsymbol{y}(\boldsymbol{s}) \leq \boldsymbol{d}$ but there does not exist a $\boldsymbol{y}' \geq \boldsymbol{y}(s)$ such that $\boldsymbol{y}' \in \mathcal{Y}$. To the best of our understanding, this is the only type of state that is included in our superset, but not in $\mathcal{S}^{\max}_{\text{knap}}$.

We must also consider the construction of our action sets $\mathcal{A}_{\text{knap}}(s)$, and we again do this by constructing a superset of this via dynamic programming, i.e., constructing partial spaces and putting them back together. To do this, we need the notion of "padding", whereby a partial action is embedded into the domain of the full action space, $\operatorname{Dom}\mathcal{A} = \bigtimes_{i=1}^{I} \mathbb{Z}_{\geq 0}^{J_i}$, by taking the 0 action across all subsystems and component types not considered by the partial action. To help define this, let $\boldsymbol{e}_{i,j}$ be the element of $\operatorname{Dom}\mathcal{A}$ which is zero everywhere except index $(i,j)$, where it is 1. We begin by constructing the single-subsystem single-type action spaces $\mathcal{A}'_{i,j}(\boldsymbol{s})$, which will be a superset of $\mathcal{A}_{i,j}(\boldsymbol{s}) := \{[\boldsymbol{a}]_{i,j} : \boldsymbol{a} \in \mathcal{A}(\boldsymbol{s})\}$. Define $\text{pad}_{i,j} : \mathbb{Z}_{\geq 0} \to \operatorname{Dom}\mathcal{A}$ as $\text{pad}_{i,j}(a) = a \cdot e_{i,j}$. We set $\mathcal{A}'_{i,j}(\boldsymbol{s}) := \left\{a_{i,j} \in \mathbb{Z} : C\boldsymbol{y}(\boldsymbol{s} \oplus \text{pad}_{i,j}(a_{i,j})) \leq \boldsymbol{d}\right\}$. This is easy to construct; simply check values $a_{i,j} = 0, 1, ...$ up until the constraint is no longer met. This allows for the case that $\boldsymbol{s} \oplus \text{pad}_{i,j}(a_{i,j})$ is only quasi-feasible, but this is easier to check. We then recursively define sets:

$$\begin{aligned}
&\mathcal{A}'_{i,1:1}(\boldsymbol{s}) := \mathcal{A}'_{i,1}(\boldsymbol{s}),\\
&\mathcal{A}'_{i,1:j}(\boldsymbol{s}) := \left\{\boldsymbol{a}_{i,1:j} \in \mathcal{A}'_{i,1:j-1} \times \mathcal{A}'_{i,j} : C\boldsymbol{y}(\boldsymbol{s} \oplus \text{pad}_{i,1:j}(\boldsymbol{a}_{i,1:j})) \leq \boldsymbol{d}\right\},
\end{aligned}$$

where $\text{pad}_{i,1:j} : \mathbb{Z}_{\geq 0}^{j} \to \operatorname{Dom}\mathcal{A}$, $\text{pad}_{i,1:j}(a_{i,1:j}) = \sum_{j'=1}^{j} [a_{i,1:j}]'_j e_{i,j'}$. Defining $\mathcal{A}'_{i,\text{knap}}(\boldsymbol{s}) := \mathcal{A}'_{i,1:J_i}(\boldsymbol{s})$ then gives supersets of the subsystem partial actions $\mathcal{A}_{i,\text{knap}} = \{[\boldsymbol{a}]_i : \boldsymbol{a} \in \mathcal{A}_{\text{knap}}(\boldsymbol{s})\}$. In turn, we use these to construct a superset of the full action space as follows:

$$\begin{aligned}
&\mathcal{A}'_{1:1}(\boldsymbol{s}) := \mathcal{A}'_{1,\text{knap}}(\boldsymbol{s}),\\
&\mathcal{A}'_{1:i}(\boldsymbol{s}) := \left\{\boldsymbol{a}_{1:i} \in \mathcal{A}'_{1:i-1}(\boldsymbol{s}) \times \mathcal{A}'_{i,\text{knap}}(\boldsymbol{s}) : C\boldsymbol{y}(\boldsymbol{s} \oplus \text{pad}_{1:i}(\boldsymbol{a}_{1:i})) \leq \boldsymbol{d}\right\},
\end{aligned}$$

where $\text{pad}_{1:i} : \left(\bigtimes_{i'=1}^{i} \mathbb{Z}_{\geq 0}^{J_{i'}}\right) \to \operatorname{Dom}\mathcal{A}$, $\text{pad}_{1:i}(a_{1:i}) = \sum_{i'=1}^{i} \sum_{j=1}^{J_{i'}} [a_{1:i}]_{i',j} e_{i',j}$. $\mathcal{A}'_{\text{knap}}(\boldsymbol{s}) := \mathcal{A}'_{1:I}(\boldsymbol{s})$ then gives our superset of $\mathcal{A}_{\text{knap}}(s)$.

The construction of a superset of $\mathcal{SA}^{\max}_{\text{knap}}$ is far more straightforward: we simply define $\mathcal{SA}^{\max'}_{\text{knap}} := \left\{(\boldsymbol{s}, \boldsymbol{a}) : \boldsymbol{s} \in \mathcal{S}^{\max'}_{\text{knap}}, \boldsymbol{a} \in \mathcal{A}'_{\text{knap}}(\boldsymbol{s})\right\}$, and construct this explicitly by

first building $\mathcal{S}^{\max\prime}_{\text{knap}}$, then iterating over this set and constructing $\mathcal{A}'_{\text{knap}}(\boldsymbol{s})$ for each $\boldsymbol{s}$.

Recall that all of this is done to enumerate all of the feasible state-action frequencies without ever having to explicitly enumerate the unconstrained (other than by $K^{\max}_{i,j}$) state space $\mathcal{S}^{\max}$ in the first place, allowing us to solve far more problems using a MILP solver than otherwise possible, especially those with higher dimensionality (i.e., number of subsystems or number of component types) but tight knapsack constraints. This in turn allows us to solve more instances to optimality than otherwise possible; however, as we will later confirm in Section 6.4, problem instances still become very computationally intensive very quickly. The remainder of this section works towards a far more scalable solution methodology.

### 6.3.2 Issues with Approximate Pareto Population

In Chapter 4, we developed the Approximate Pareto Population (APP) methodology, whereby a Pareto front of candidate designs is identified based on simplifying assumptions on the policy used in the MDP, and then the exact Pareto front of policies is found for each design. For the parallel-system, the simple policy for any design just repaired any component as soon as it became damaged, and this yielded an initially non-linear design problem which in turn was linearised using probability chains.

Our success with APP in Chapter 4 suggests that it may be a good approach for the series-parallel case. Indeed, it remains true that a non-linear design problem can be formulated by assuming that any installed component will always be repaired straight away. For a fixed design, the MDP would then be solvable via the decomposition method used in Chapter 5. However, to the best of our knowledge, the design problem for series-parallel systems as we've described cannot be efficiently linearised, either by probability chains or by any other method. The resulting series-parallel design problem is far closer to the standard series-parallel RAP as seen in the literature. The most common class of solution methodologies applied here are metaheuristics such as NSGA-II; however,

these do not have guarantees of optimality or completeness of the Pareto front, and they lie outside the scope of this work.

As an alternative, the work of Zia and Coit (2010) demonstrated that Dantzig-Wolfe decomposition and column generation could be used to approximately solve a series-parallel RAP, and Reihaneh et al. (2022) extended this to an exact branch-and-price algorithm. We previously applied Dantzig-Wolfe decomposition to decomposable impulsive CTMDPs in Chapter 5, and it seems reasonable to apply it here. Of course, the heuristic of Zia and Coit (2010) would also not have guarantees on optimality, and the branch-and-price of Reihaneh et al. (2022) would have to be adapted to the usage costs case. However, the use of Dantzig-Wolfe begs the question: if we would apply Dantzig-Wolfe decomposition to the design-only problem in SP1 of APP, why would we not instead apply it to the whole problem? For an $I$ subsystem problem, we would roughly expect the system to decompose into $I$ subproblems to be repeatedly resolved for the purpose of column generation, where these subproblems would be similar to the parallel system problems of Chapter 4. Additionally, we already know that APP works well for those problems, whereas applying APP to the SPIDMP directly may yield disappointing results, considering all the work that would be required in solving the design problem. As such, the remainder of this section is dedicated to the use of Dantzig-Wolfe decomposition for BO-SPIDMP.

### 6.3.3 Block-Diagonalization and Decomposition

Along similar lines to Chapter 5, SPIDMP can be approximately decomposed by assuming the use of a coherent separable policy, and in turn block-diagonalising the DI-CTMDP and applying Dantzig-Wolfe decomposition to obtain a problem compatible with column generation techniques. However, unlike the non-Design problem in that chapter, we must account for variables with linking knapsack constraints in SPIDMP. The additional assumptions and block-diagonalisation yield an augmented problem that

we call the *Decomposed Coherent SPIDMP*, or DC-SPIDMP:

$$\text{(DC-SPIDMP)} \quad \min_{\boldsymbol{x},\boldsymbol{\pi},\boldsymbol{\eta}} f^o\left(\left(\sum_{(\boldsymbol{s}_i,\boldsymbol{a}_i)\in\mathcal{SA}^{\max}_{i,\text{knap}}} c_i^o(\boldsymbol{s}_i\oplus\boldsymbol{a}_i)\pi_i(\boldsymbol{s}_i,\boldsymbol{a}_i)\right)_{i=1}^{I}\right)$$

$$\min_{\boldsymbol{x},\boldsymbol{\pi},\boldsymbol{\eta}} f^f\left(\left(\sum_{(\boldsymbol{s}_i,\boldsymbol{a}_i)\in\mathcal{SA}^{\max}_{i,\text{knap}}} c_i^f(\boldsymbol{s}_i\oplus\boldsymbol{a}_i)\pi_i(\boldsymbol{s}_i,\boldsymbol{a}_i)\right)_{i=1}^{I}\right)$$

$$\text{s.t. } C'\boldsymbol{x}\le\boldsymbol{d}, \tag{6.3.1}$$

for each $i\in\{1,...,I\}$ :

$$\sum_{(\boldsymbol{s}_i,\boldsymbol{a}_i)\in\mathcal{SA}^{\max}_{i,\text{knap}}} q_i(\boldsymbol{s}_i,\boldsymbol{a}_i,\boldsymbol{s}_i')\pi_i(\boldsymbol{s}_i,\boldsymbol{a}_i)=0,\ \text{for all } \boldsymbol{s}_i'\in\mathcal{S}^{\max}_{i,\text{knap}}, \tag{6.3.2}$$

$$\sum_{(\boldsymbol{s}_i,\boldsymbol{a}_i)\in\mathcal{SA}^{\max}_{i,\text{knap}}} \pi_i(\boldsymbol{s}_i,\boldsymbol{a}_i)=1,$$

$$\pi_i(\boldsymbol{s}_i,\boldsymbol{a}_i)\ge 0,\ \text{for all } (\boldsymbol{s}_i,\boldsymbol{a}_i)\in\mathcal{SA}^{\max}_{i,\text{knap}}, \tag{6.3.3}$$

$$\pi_i(\boldsymbol{s}_i,\boldsymbol{a}_i)\le\eta_i(\boldsymbol{s}_i,\boldsymbol{a}_i),\ \text{for all } (\boldsymbol{s}_i,\boldsymbol{a}_i)\in\mathcal{SA}^{\max}_{i,\text{knap}}, \tag{6.3.4}$$

$$\sum_{\boldsymbol{a}_i'\in\mathcal{A}_{i,\text{knap}}(\boldsymbol{s}_i),\boldsymbol{a}_i'\neq\boldsymbol{a}_i} \pi_i(\boldsymbol{s}_i,\boldsymbol{a}_i')\le 1-\eta_i(\boldsymbol{s}_i,\boldsymbol{a}_i),\ \text{for all } (\boldsymbol{s}_i,\boldsymbol{a}_i)\in\mathcal{SA}^{\max}_{i,\text{knap}}, \tag{6.3.5}$$

$$\sum_{\boldsymbol{a}_i'\in\mathcal{A}_{i,\text{knap}}(\boldsymbol{s}),\boldsymbol{a}_i'\neq 0} \pi_i(\boldsymbol{s}_i\oplus\boldsymbol{a}_i,\boldsymbol{a}_i')\le 1-\eta_i(\boldsymbol{s}_i,\boldsymbol{a}_i),\ \text{for all } (\boldsymbol{s}_i,\boldsymbol{a}_i)\in\mathcal{SA}^{\max}_{i,\text{knap}}, \tag{6.3.6}$$

$$\sum_{\substack{(\boldsymbol{s}_i,\boldsymbol{a}_i)\in\mathcal{SA}^{\max}_{i,\text{knap}}:\\ [\boldsymbol{s}_i\oplus_i\boldsymbol{a}_i]_{j,2}\le K^{\max}_{i,j}-k}} \pi_i(\boldsymbol{s}_i,\boldsymbol{a}_i)\le x_{i,j,k},\ \text{for all } j,k, \tag{6.3.7}$$

$$x_{i,j,k}-x_{i,j,k+1}\ge 0,\ \text{for all } i,j,\ \text{and } k\le K^{\max}_{i,j}-1, \tag{6.3.8}$$

$$\sum_{j=1}^{J_i}\sum_{k=1}^{K^{\max}_{i,j}} x_{i,j,k}\ge 1, \tag{6.3.9}$$

$$x_{i,j,k}\in\{0,1\},\ \text{for all } i\in[I], j\in[J_i], k\in[K^{\max}_{i,j}], \tag{6.3.10}$$

$$\eta_i(\boldsymbol{s}_i,\boldsymbol{a}_i)\in\{0,1\},\ \text{for all } i\in[I], (\boldsymbol{s}_i,\boldsymbol{a}_i)\in\mathcal{SA}^{\max}_{i,\text{knap}}. \tag{6.3.11}$$

DC-SPIDMP is a block-diagonal formulation of BO-SPIDMP under the assumption that the policy used will be coherent and separable. This formulation has separate state-action frequencies $\boldsymbol{\pi}_i$ for each of the separate subsystems, and introduces binary variables $\boldsymbol{\eta}_i$ which tell us if we use action $\boldsymbol{a}_i$ in subsystem state $\boldsymbol{s}_i$, if this state is recurrent. The two objectives are the long-run average operational costs and system failure probability respectively, as calculated from the subsystem long-run average costs and subsystem failure probabilities using the functions $f^o : \mathbb{R}^I \to \mathbb{R}, f^o(g_1^o, ..., g_I^o) = \sum_{i=1}^I g_i^o$ and $f^f(g_1^f, ..., g_I^f) = 1 - \prod_{i=1}^I (1 - g_i^f)$ for the additive and logical-OR cost functions respectively. Constraints (6.3.1) provide the linking knapsack constraints that must hold across all blocks. The remaining constraints are block-diagonal in nature, meaning they are repeated for each subsystem $i$ and each copy only includes variables for subsystem $i$. Constraints (6.3.2)-(6.3.3) ensure that the state-action frequencies for the $i^{\text{th}}$ subsystem satisfy the CTMDP balance equations and represent a valid probability distribution. Constraint (6.3.4) ensures that $\pi_i(\boldsymbol{s}_i, \boldsymbol{a}_i)$ can only be non-zero if $\eta_i(\boldsymbol{s}_i, \boldsymbol{a}_i)$ is 1. Constraint (6.3.5) ensures that the policy is deterministic, so if $\eta_i(\boldsymbol{s}_i, \boldsymbol{a}_i) = 1$ for some state action pair, then $\pi_i(\boldsymbol{s}_i, \boldsymbol{a}_i') = 0$ for all other state-action pairs. Constraint (6.3.6) enforces coherence, so if $\eta_i(\boldsymbol{s}_i, \boldsymbol{a}_i) = 1$ then all non-zero actions in the target state $\boldsymbol{s}_i \oplus_i \boldsymbol{a}_i$ are prohibited. Constraint (6.3.7) links the design decisions to the state-action frequencies, prohibiting non-zero state-action frequencies for state-action pairs that represent the existence of a component that has not been installed. Constraints (6.3.8) and (6.3.9) carry over the symmetry breaking constraints and ensure that there is at least one component per subsystem, respectively. Constraints (6.3.10) and (6.3.11) ensures the $\boldsymbol{x}$ and $\boldsymbol{\eta}$ variables are binary.

We can see that every solution (of which there are finitely many) to DC-SPIDMP corresponds to an indexed set of design-policy pairs $\{(\boldsymbol{x}_i, \mu_i)\}_{i=1}^I$, where for each subsystem $i$, $\boldsymbol{x}_i = \left((x_{i,j,k})_{k=1}^{K_{i,j}^{\max}}\right)_{j=1}^{J_i}$ gives the design decisions and $\mu_i$ is a coherent policy. The block diagonal nature of this formulation allows us to use Dantzig-Wolfe decompo-

sition, whereby the $\boldsymbol{x}, \boldsymbol{\pi}$ and $\boldsymbol{\eta}$ decision variables are replaced by so-called *disaggregated* variables $\lambda_i^{(\boldsymbol{x}_i,\mu_i)} \in \{0,1\}$, each of which corresponds to a subsystem design-policy pair $(\boldsymbol{x}_i, \mu_i)$. For each $i$, we let:

$$\mathcal{X}_i = \left\{ x_i \in \bigtimes_{j=1}^{J_i} \bigtimes_{k=1}^{K_{i,j}^{\max}} \{0,1\} : x_i \text{ is feasible for DC-SPIDMP} \right\},$$

be the set of all feasible $i^{\text{th}}$ subsystem design solutions to DC-SPIDMP (namely, those that obey the knapsack and symmetry breaking constraints, leaving enough knapsack room for the other subsystems to have at least one component each), and let $\mathcal{M}_i := \left\{\mu_i : \mathcal{S}_{i,\text{knap}}^{\max} \to \mathcal{A}_{i,\text{knap}} \text{ s.t. } \mu_i \text{ is coherent}\right\}$ be the unconstrained set of coherent policies of the $i^{\text{th}}$ subsystem. Then, we define

$$\mathcal{M}_i(\boldsymbol{x}_i) := \left\{\mu_i \in \mathcal{M}_i \text{ s.t. } \mu_i(\boldsymbol{s}_i) = \boldsymbol{a}_i \text{ only if } [\boldsymbol{s}_i \oplus_i \boldsymbol{a}_i]_{j,2} \geq K_{i,j}^{\max} - k(1 - x_{i,j,k}) \text{ for all } j,k\right\},$$

to be the set of coherent policies for subsystem $i$ which are feasible with respect to design $x_i$ (i.e., does not repair more of any component than those actually installed), and then define $\mathcal{XM}_i := \{(\boldsymbol{x}_i, \mu_i) : \boldsymbol{x}_i \in \mathcal{X}_i, \mu_i \in \mathcal{M}_i(\boldsymbol{x}_i)\}$ to be the set of all feasible design-policy pairs for subsystem $i$ in DC-SPIDMP. For each subsystem, we then have a set of disaggregated decision variables $\Lambda_i = \left\{\lambda_i^{(\boldsymbol{x}_i,\mu_i)} : (\boldsymbol{x}_i, \mu_i) \in \mathcal{XM}_i\right\}$. We then define a new knapsack cost matrix $C^\Lambda$ such that $\left[C^\Lambda\right]_{l,(\boldsymbol{x}_i,\mu_i)} = \sum_{j=1}^{J_i} \sum_{k=1}^{K_{i,j}^{\max}} C_{i,j,l} x_{i,j,k}$. For non-empty subsets $\overline{\mathcal{XM}}_i \subset \mathcal{XM}_i$ of the decision variable indices, and the corresponding subsets of decision variables $\bar{\Lambda}_i \subset \Lambda_i$, and applying the necessary linearising transformation to the second objective as per Chapter 5, we can then define the SPIDMP *reduced master problem*, or SPIDMP-RMP:

$$\text{(SPIDMP-RMP)} \quad \min_{\boldsymbol{\lambda}} \sum_{i=1}^{I} \sum_{(\boldsymbol{x}_i,\mu_i)\in\overline{\mathcal{XM}}_i} g_i^o(\mu_i)\lambda_i^{(\boldsymbol{x}_i,\mu_i)} \tag{6.3.12}$$

$$\min_{\boldsymbol{\lambda}} \sum_{i=1}^{I} \sum_{(\boldsymbol{x}_i,\mu_i)\in\overline{\mathcal{XM}}_i} -\ln\left\{1-g_i^f(\mu_i)\right\}\lambda_i^{(\boldsymbol{x}_i,\mu_i)} \tag{6.3.13}$$

$$\text{s.t. } C^{\Lambda}\boldsymbol{\lambda} \leq \boldsymbol{d}, \tag{6.3.14}$$

$$\text{for all } i \in \{1,...,I\}:$$

$$\sum_{(\boldsymbol{x}_i,\mu_i)\in\overline{\mathcal{XM}}_i} \lambda_i^{(\boldsymbol{x}_i,\mu_i)} = 1, \tag{6.3.15}$$

$$\lambda_i^{(\boldsymbol{x}_i,\mu_i)} \in \{0,1\}, \text{ for all } (\boldsymbol{x}_i,\mu_i) \in \overline{\mathcal{XM}}_i. \tag{6.3.16}$$

If $\overline{\mathcal{XM}}_i = \mathcal{XM}_i$ for all $i$ then we call this the master problem, or SPIDMP-MP, which of course has an exponentially large number of decision variables. Additionally, we call this problem with only the first (cost) objective SPIDMP-(R)MP-C, and the problem with only the second (failure) objectives is called SPIDMP-(R)MP-F. Objectives (6.3.12) and (6.3.13) are the disaggregated cost and failure probability objectives, respectively, with the second objective linearised. Constraint (6.3.14) give the disaggregated knapsack constraints, constraint (6.3.15) ensures that only one solution is chosen per block/subsystem, and constraint (6.3.16) ensures that the $\lambda$ variables are binary.

Solving SPIDMP-MP is equivalent to solving DC-SPIDMP, which in turn is equivalent to finding heuristic solutions (i.e., limited to coherent separable policies) to BO-SPIDMP. The goal now is to (approximately) solve SPIDMP-MP by finding subsets $\overline{\mathcal{XM}}_i \subset \mathcal{XM}_i$ such that SPIDMP-RMP and SPIDMP-MP have the similar objective Pareto-fronts. Two big sources of the combinatorial nature of $\mathcal{XM}_i$ are that it allows for component-mixing, meaning it needs to consider every possible combination of components, and that for every design it includes every possible policy. We now follow with a strategy of enumerating all one-component-type solutions along with their

Pareto-optimal policies, and then follow with an iterative procedure for generating new design-policy pairs which does allow for component-mixing.

### 6.3.4 Solving Without Component-Mixing

By imposing no component-mixing, we are imposing the following set of constraints on DC-SPIDMP:

$$\sum_{j=1}^{J_i} x_{i,j,1} = 1, \text{ for all } i.$$

In theory, with these constraints included, DC-SPIDMP can be solved to true Pareto-optimality by fully enumerating all design-policy pairs for each subsystem which correspond to a Pareto-optimal policy for any fixed design, and adding these to $\overline{\mathcal{XM}_i}$. However, it's important to note a caveat for this.

**Remark 6.3.1.** *In practice, the fixed-design subsystem MDPs will be solved using the dichotomic method, meaning some policies that are optimal for the same objective weightings could be missed (however this is unlikely), and those solutions that are Pareto-optimal with respective to other DS policies but Pareto-dominated by a stochastic policy will not be found. Potentially, it could be the case that a policy for the whole system that is Pareto-optimal amongst coherent stationary policies could use one of these stochastically-dominated policies as one of its subpolicies. We have not disproved this possibility.*

Using the single-type subsystem solutions generated, SPIDMP-RMP can then be solved using the $\varepsilon$-constraint method. While the initial subsystem solution generation process does involve solving a lot of bi-objective CTMDPs, our prior experience suggests that these single-subsystem one-component-type problems can be solved very quickly using the dichotomic method and an LP solver. Algorithm 1 provides the algorithm for this method. Note that step 3 is listed as optional, as it may be the case that a given solver will do some pre-processing to ignore the dominated solutions.

**Algorithm 1** SPIDMP-MP Solver (No Mixing)

1. Initialise: $\overline{\mathcal{XM}_i} = \emptyset$ for all $i \in [I]$.
2. Generate columns: For $i \in [I], j \in [J_i], k \in [K_{i,j}^{\max}]$:
   - Solve single-subsystem bi-objective maintenance MDP with $k$ copies of component type $j$.
   - For each Pareto-optimal policy found (except the never repair policy), add corresponding design-policy pair to $\overline{\mathcal{XM}_i}$ and record objective values and constraint contributions.
3. (Optional) Remove dominated solutions: For each subsystem $i$ and design-policy pair $(\boldsymbol{x}_i, \mu_i)$, check over all other solutions for this subsystem; if there exists a solution that dominates it across both objectives and all constraints, eliminate it from $\overline{\mathcal{XM}_i}$.
4. Reassemble solutions: solve SPIDMP-RMP with solutions sets $\overline{\mathcal{XM}_i}$ using a MILP solver. Solve single-objective reliability-only problem first, then find the rest of the Pareto front by using $\varepsilon$-constraint method (start by solving with $\varepsilon = 0$, then repeatedly update $\varepsilon$ so it "cuts off" previous solution and re-solve, and do this until $\varepsilon$ would exceed the optimal reliability value).

Solutions found using this algorithm will be near Pareto-optimal for DC-SPIDMP (up to the sensitivity on updating $\varepsilon$) under the additional no-mixing constraints, providing a strong baseline. These solutions can then be improved upon by allowing for component-mixing (unless this was forbidden in the original problem), and we do this by performing column generation via *pricing subproblems* rather than complete enumeration. The following subsection explores this approach.

### 6.3.5 Solving With Component-Mixing

*Note: This subsection, and the remainder of Section 6.3, focuses on ideas about how to approximately solve the problem at hand. However, they are not implemented nor tested in the computational study.*

Given a feasible set of starting variables, such as those given in the previous subsection, the process of iteratively adding new variables is a method known as *column*

*generation*, which identifies variables to be added to the RMP by solving a scalarisation of its linear relaxation, and uses the dual variables from the optimal solution of that problem to formulate a *pricing subproblem*. This is a smaller optimisation problem that identifies a promising $(\boldsymbol{x}_i, \mu_i)$ pair to be added to the subset. However, in order to obtain such dual variables to define a pricing subproblem, we must work with single-objective continuous relaxations of SPIDMP-RMP. We begin by considering the easier case of solving the continuous relaxation of SPIDMP-MP-F (or CR-SPIDMP-MP-F), and then move onto solving the continuous relaxation of a mixed-objective version of SPIDMP-MP. Due to the abundance of problem variants and subproblems in this methodology, all of which are named, this subsection concludes with a table of acronyms, full names, and purposes for each.

**Failure-Only Minimisation**

We first focus on solving CR-SPIDMP-MP-F via column generation, i.e., by iteratively adding columns to CR-SPIDMP-RMP-F. For any column sets $\left\{\overline{\mathcal{XM}_i}\right\}_{i\in[I]}$, the latter problem has dual variables $\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_L)$, one for each knapsack constraint, and $\boldsymbol{\beta} = (\beta_1, ..., \beta_I)$, one for each subsystem sum-to-one constraint. In this case, for each subsystem (or block) we get reduced cost function:

$$rc^f\left(\lambda_i^{(\boldsymbol{x}_i,\mu_i)}\right) := -\ln\left\{1 - g_i^f(\mu_i)\right\} - \boldsymbol{\sigma}^T\left[C'\right]_i \boldsymbol{x}_i - \beta_i,$$

where $\left[C'\right]_i$ is the submatrix of $C'$ containing all rows but only the columns pertaining to subsystem $i$. The minimisation of this reduced cost function does not require dynamic maintenance optimisation, as the policy that minimises failure probability with no concern for operational cost is of course the policy that repairs every component as soon as it becomes damaged, which for any given design we may denote by $\mu_i^{\mathrm{FA}}(\boldsymbol{x}_i)$. The problem of minimising this reduced cost, which we call the *Knapsack Penalised*

*Subsystem Design Problem*, or KPSDP, is then given as follows:

$$\text{(KPSDP)} \quad \min_{\boldsymbol{x}_i \in \mathcal{X}_i} -\ln\left\{1 - g_i^f\left(\mu_i^{\text{FA}}(\boldsymbol{x}_i)\right)\right\} - \boldsymbol{\sigma}^T \left[C'\right]_i \boldsymbol{x}_i - \beta_i.$$

Despite the explicit nonlinearity of the log term, we now want to demonstrate that this problem can be solved via mixed integer linear programming. First, we make the objective more explicit by expressing $g_i^f\left(\mu_i^{\text{FA}}(\boldsymbol{x}_i)\right)$ in closed form. We define $p_{i,j} := \tau_{i,j}/(\tau_{i,j} + \alpha_{i,j})$ to be the long-run average probability that component type $j$ in subsystem $i$ is operational under a fully active maintenance policy, and $y_{i,j} = \sum_{k=1}^{K_{i,j}^{\max}} x_{i,j,k}$ to be the number of copies of component type $j$ in subsystem $i$ being installed. Then we have:

$$g_i^f\left(\mu_i^{\text{FA}}(\boldsymbol{x}_i)\right) = \prod_{j=1}^{J_i} (1 - p_{i,j})^{y_{i,j}}.$$

As such, plugging this into KPSDP yields a nonlinear program. We can avoid this by transforming the problem into a bi-objective program and linearising the suitable objective. Optimal solutions to KPSDP will be Pareto-optimal solutions to the bi-objective problem:

$$\min_{\boldsymbol{x}_i \in \mathcal{X}_i} -\ln\left\{1 - g_i^f\left(\mu_i^{\text{FA}}(\boldsymbol{x}_i)\right)\right\}$$
$$\min_{\boldsymbol{x}_i \in \mathcal{X}_i} -\boldsymbol{\sigma}_L^T \left[C'\right]_i \boldsymbol{x}_i,$$

which, if we apply the strictly increasing transformation $T(z) = \ln(1 - e^{-z})$ to the first objective, expand out the second, and replace the $x$ variables with general integer $y$ variables (with space $\mathcal{Y}_i$ that carries over the feasibility constraints of $\mathcal{X}_i$), yields the

bi-objective integer linear program:

$$\min_{\boldsymbol{y}_i \in \mathcal{Y}_i} \sum_{j=1}^{J_i} \ln(1-p_{i,j}) y_{i,j}$$
$$\min_{\boldsymbol{y}_i \in \mathcal{Y}_i} \sum_{l=1}^{L} |\sigma_l| \sum_{j=1}^{J_i} C_{i,j,l} y_{i,j},$$

noting that the $\sigma_l$ are all non-positive due to being dual variables for a less-than constraint in a minimisation problem. Finally, we apply the $\varepsilon$-constraint method to the second objective, and represent the constraints of the $\mathcal{Y}_i$ set explicitly, yielding the *$\varepsilon$-Knapsack-Constrained Subsystem Design Problem*, or $\varepsilon$-KCSDP:

$$\begin{aligned}
(\varepsilon\text{-KCSDP}) \quad \min_{\boldsymbol{y}} \ & \sum_{j=1}^{J_i} \ln(1-p_{i,j}) y_{i,j} \\
\text{s.t. } & C\boldsymbol{y} \leq \boldsymbol{d}, \\
& \sum_{l=1}^{L} |\sigma_l| \sum_{j=1}^{J_i} C_{i,j,l} y_{i,j} \leq \varepsilon, \\
& \sum_{j=1}^{J_i} y_{i,j} \geq 1, \text{ for all } i = 1, ..., I, \\
& y_{i,j} \in \mathbb{Z}_{\geq 0}, \text{ for all } i, j.
\end{aligned}$$

Note that we include the whole-system design variables and constraints explicitly to prevent the generation of infeasible columns. To generate a column for subsystem $i$, we enumerate the whole Pareto-front of the bi-objective problem using the $\varepsilon$-constraint method, and test each solution against the KPSDP objective function to find the minimiser.

**Remark 6.3.2.** *For the purpose of column generation, rather than finding the whole front, it could be enough to solve for different values of $\varepsilon$ until either: (i) the first solution with negative reduced cost is found, or (ii) continue resolving to obtain better reduced cost until an update in $\varepsilon$ leads to a worst reduced cost than the best negative*

*reduced cost found so far. The latter can be thought of as giving up upon finding an "inflection point" for the objective of KPSDP in terms of $\varepsilon$ (however, this is just a heuristic for early termination of the $\varepsilon$-constraint procedure; we do not suggest or prove that KPSDP is in any way "convex in $\varepsilon$"). This would result in faster generation of columns, but could require more iterations of column generation. Such early termination is only worth considering if generating the whole front is found to be slow.*

Repeatedly adding columns to CR-SPIDMP-RMP-F by solving KPSDP via the surrogate problem $\varepsilon$-KCSDP (for each subsystem) yields an optimal solution to CR-SPIDMP-MP-F. In turn, the columns generated can be used in the integral version of SPIDMP-RMP to obtain heuristic solutions that hopefully improve upon the no-mixing solutions. It is worth asking what should be done with the other designs generated by $\varepsilon$-KCSDP. Adding these to the RMP will allow for more flexibility in the generation of integral solutions, and they all strike some sort of balance between reliability and constraints. On the other hand, doing this every time could add too many binary variables to the problem, worsening runtimes. The same can also be asked of the Pareto fronts of maintenance policies for all such designs: in principle, for every design generated, we could solve the respective maintenance MDP and add these columns to be considered by the bi-objective RMP, but this would of course add a computational burden as well as a greatly increased number of binary variables. Algorithm 2 provides the algorithm for solving KPSDP (the pricing subproblem), and Algorithm 3 provides the algorithm for solving for CR-SPIDMP-MP-F and returning the generated columns. We now follow with the solution methodology for instances of the mixed-objective problem.

**Algorithm 2** KPSDP Solver

1. Input: Subsystem parameters $\mathcal{P}_i$, dual variables $\sigma$ and $\beta$.
2. Initialise solution set $\mathcal{Y}^* = \emptyset$.
3. Set $\varepsilon_{\min} = \min_j \left\{ \sum_{l=1}^{L} |\sigma_L| C_{i,j,l} \right\}$.
4. Set $\varepsilon = \infty$. Repeat until $\varepsilon < \varepsilon_{\min}$:
   - Solve $\varepsilon$-KCSDP, obtaining solution $\boldsymbol{y}_\varepsilon^*$.
   - Add $\boldsymbol{y}_\varepsilon^*$ to $\mathcal{Y}^*$, set $\varepsilon = \text{Dec}\left( \sum_{l=1}^{L} |\sigma_l| \sum_{j=1}^{J_i} C_{i,j,l} [y_\varepsilon^*]_{i,j} \right)$.
5. Iterate over $\mathcal{Y}^*$ to find the solution that minimises KPSDP. Return this solution in binary form.

**Algorithm 3** CR-SPIDMP-MP-F Solver

1. Input: Problem parameters $\mathcal{P}$.
2. Initialise $\overline{\mathcal{XM}_i}$ for each $i$ using steps 1-3 of Algorithm 1. Solve CR-SPIDMP-RMP-F with these initial columns using an LP solver, obtaining dual variables $\boldsymbol{\sigma}$ and $\boldsymbol{\beta}$.
3. Set $rc_i = -\infty$ for each $i$. Repeat until $rc_i \geq 0$ for all $i$:
   - For each $i$, solve KPSDP for subsystem $i$ with dual variables $\boldsymbol{\sigma}, \boldsymbol{\beta}$ using Algorithm 2, and record the KPSDP objective value as $rc_i$. If $rc_i < 0$, add this new column to $\overline{\mathcal{XM}_i}$.
   - If a new column was found: re-solve CR-SPIDMP-RMP-F with the updated column sets and update the dual variables $\boldsymbol{\sigma}, \boldsymbol{\beta}$.
4. Return the column sets $\left\{ \overline{\mathcal{XM}_i} \right\}_{i \in [I]}$.

**Mixed-Objective Minimisation**

We choose to focus on the mixed-objective scalarisation CR-SPIDMP-RMP-P for two reasons: it can be integrated into a dichotomic search; and as we're only ever solving the continuous relaxation of SPIDMP-MP, we need not worry about unsupported solutions. This scalarisation yields the problem SPIDMP-RMP-P:

$$\begin{aligned}\text{(CR-SPIDMP-RMP-P)} \quad \min_{\boldsymbol{\lambda}} & \sum_{i=1}^{I} \sum_{(\boldsymbol{x}_i,\mu_i)\in\overline{\mathcal{XM}}_i} \left(g_i^o(\mu_i) - P\ln\left\{1-g_i^f(\mu_i)\right\}\right)\lambda_i^{(\boldsymbol{x}_i,\mu_i)} \\ \text{s.t. } & \text{(6.3.14)} - \text{(6.3.15)}, \\ & \lambda_i^{(\boldsymbol{x}_i,\mu_i)} \in [0,1], \text{ for all } (\boldsymbol{x}_i,\mu_i) \in \overline{\mathcal{XM}}_i, \text{ for all } i,\end{aligned}$$

where $P$ is the weight (or penalty) for the second objective. Now, when we solve CR-SPIDMP-RMP-P, we again obtain dual variables $\sigma_l$ for each of the knapsack constraints, and $\beta_i$ for each of the sum-to-one constraints. Then, for any variable $\lambda_i^{(\boldsymbol{x}_i,\mu_i)}$, we can define the *reduced cost* as follows:

$$rc\left(\lambda_i^{(\boldsymbol{x}_i,\mu_i)}\right) := g_i^o(\mu_i) - P\ln\left\{1-g_i^f(\mu_i)\right\} - \boldsymbol{\sigma}^T\left[C'\right]_i\boldsymbol{x}_i - \beta_i,$$

where $[C']_i$ is a submatrix containing all rows of $C'$ but only the columns that pertain to the $i^{\text{th}}$ subsystem. The problem of finding a subsystem design-policy pair with the most negative reduced cost is then itself an MDP Design problem which we call the *Knapsack Penalised Subsystem Design and Maintenance Problem*, or KPSDMP:

$$\text{(KPSDMP)} \quad \min_{(\boldsymbol{x}_i,\mu_i)\in\mathcal{XM}_i} \left\{g_i^o(\mu_i) - P\ln\left\{1-g_i^f(\mu_i)\right\} - \boldsymbol{\sigma}^T\left[C'\right]_i\boldsymbol{x}_i - \beta_i\right\}.$$

KPSDMP itself is an interesting problem, as it is an MDP Design problem with a nonlinear objective function, and as such solving it exactly is out of the scope of this

work. Instead, we will need to investigate heuristic solutions. Along similar lines to KPSDP, we can recognise that an optimal solution of KPSDMP belongs to the Pareto front of the following bi-objective problem:

$$\min_{(\boldsymbol{x}_i,\mu_i)\in\mathcal{XM}_i} \left\{ g_i^o(\mu_i) - \boldsymbol{\sigma}^T [C']_i \boldsymbol{x}_i \right\}$$
$$\min_{(\boldsymbol{x}_i,\mu_i)\in\mathcal{XM}_i} \left\{ g_i^f(\mu_i) \right\}.$$

We call this problem the *Bi-Objective Knapsack-Penalised Subsystem Design and Maintenance Problem*, or BO-KPSDMP. This is a bi-objective MDP Design problem which bears a great resemblance to the BO-IDDMP of Chapter 4, which is what we hoped for when deciding to pursue a decomposition method for BO-SPIDMP. Specifically, this is an MDP Design problem with quite relaxed constraints on the feasible design space (indeed, any design that allows for one component in each of the other subsystems is valid). As such, the maximal state-action space for this problem gets very large as the number of component types $J_i$ increases, meaning we have to solve this problem heuristically. We will again use the APP methodology that we developed in Chapter 4. For the design phase (SP1) of APP, we again choose to use the heuristic that always repairs components as soon as they fail. The resulting design problem maintains the non-linearities from the usage costs and failure objective that we encountered in BO-DOP in Chapter 4, and we therefore again apply the probability chains method for linearisation (requiring the addition of a small penalty term for system failure within the operational cost objective), along with applying the $\varepsilon$-constraint method to the failure objective to scalarise the problem and linearise that term. We call the resulting problem the *$\varepsilon$-constrained $\delta$-augmented knapsack-penalised subsystem design-only problem*, or $\varepsilon$-$\delta$-KPSDOP, which assuming $c_{i,1} \leq c_{i,2} \leq \cdots \leq c_{i,J_i}$ can be formulated as follows:

$$
\begin{aligned}
(\varepsilon\text{-}\delta\text{-KPSDOP}) \quad \min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}} & \sum_{j=1}^{J_i} \sum_{k=1}^{K_{i,j}^{\max}} \left\{ \left( r_{i,j} q_{i,j} + \sum_{l=1}^{L} \sigma_l C_{i,j,l} \right) x_{i,j,k} + c_{i,j} y_{j,k} \right\} \\
& + (1+\delta) c_{i,J_i} z_{J_i, K_{i,j}^{\max}} && (6.3.17) \\
\text{s.t. } & C'\boldsymbol{x} \leq \boldsymbol{d}, && (6.3.18) \\
& \sum_{j=1}^{J_{i'}} x_{i',j,1} = 1, \text{ for all } i' \in [I], && (6.3.19) \\
& x_{i,j,k+1} \leq x_{i,j,k}, \text{ for all } j \in [J_i], k \in [K_{i,j}^{\max}], && (6.3.20) \\
& y_{1,1} + z_{1,1} = 1, && (6.3.21) \\
& y_{j,k+1} + z_{j,k+1} = z_{j,k}, \text{ for all } j \in [J_i], k \in [K_{i,j}^{\max} - 1], && (6.3.22) \\
& y_{j,1} + z_{j,1} = z_{j-1,K_{i,j-1}^{\max}}, \text{ for all } j = 2, \ldots, J_i, && (6.3.23) \\
& y_{j,k} \leq p_j x_{i,j,k}, \text{ for all } j \in [J_i], k \in [K_{i,j}^{\max}], && (6.3.24) \\
& y_{j,k+1} \leq p_j z_{j,k}. \text{ for all } j \in [J_i], k = [K_{i,j}^{\max}], && (6.3.25) \\
& y_{j,1} \leq p_j z_{j-1,K_{i,j-1}^{\max}}, && (6.3.26) \\
& \sum_{j=1}^{J_i} \sum_{k=1}^{K_{i,j}^{\max}} \ln(q_{i,j}) x_{i,j,k} \leq \varepsilon., && (6.3.27) \\
& x_{i,j,k} \in \{0,1\}, \text{ for all } j \in [J_i], k \in [K_{i,j}^{\max}] \\
& x_{i',j,1} \in \{0,1\}, \text{ for all } i' \neq i, j \in [J_{i'}], \\
& y_{j,k}, z_{j,k} \geq 0, \text{ for all } j \in [J_i], k \in [K_{i,j}^{\max}].
\end{aligned}
$$

The variables $x$ range over every valid component type $j$ and copy index $k$ for subsystem $i$, but only allow for one copy of each component type $i'$ in the other subsystems (this just ensures that this subsystem solution is feasible for the whole subsystem). Variables $\boldsymbol{y}$ and $\boldsymbol{z}$ are used for the probability chains linearisation (note that only in the context of probability chains does $\boldsymbol{y}$ not refer to integer design variables). Objective (6.3.17) gives the $\delta$-augmented objective function, containing repair costs and dual

costs in terms of design variables $\boldsymbol{x}$, usage costs in terms of the linearising variables $\boldsymbol{y}$, and a small penalty for failure in terms of $\boldsymbol{z}$. This small penalty shouldn't affect the optimal solution in most cases; it will only have an effect when the most expensive usage cost is orders of magnitude greater than the cheapest usage cost (otherwise, the $\varepsilon$-constraint keeps this term very small). Constraint (6.3.18) ensures the design variables are feasible for the whole system. Constraint (6.3.19) ensures that each subsystem would have at least one component. Constraint (6.3.20) are symmetry breaking constraints. Constraints (6.3.21), (6.3.22), (6.3.23), (6.3.24), (6.3.25), (6.3.26) are the probability chain constraints, as introduced in Chapter 4. Constraint (6.3.27) gives the linearised $\varepsilon-$constraint. The remaining constraints define the domains of the variables.

The problem $\varepsilon$-$\delta$-KPSDOP is of course only feasible down to a minimum achievable value of $\varepsilon$. To determine this value, we use another problem called the *Failure-Only Subsystem Design-Only Problem*, or F-SDOP, which is analogous to F-DOP in Chapter 4. The formulation is as follows:

$$
\begin{aligned}
\text{(F-SDOP)} \quad \min_{\boldsymbol{y}} \ & \sum_{j=1}^{J_i} \ln(q_{i,j}) y_{i,j} \\
\text{s.t. } & C\boldsymbol{y} \leq \boldsymbol{d}, \\
& \sum_{j=1}^{J_i} y_{i,j} \geq 1, \\
& \sum_{j=1}^{J_{i'}} y_{i',j} = 1, \ \text{for all } i' \in [I] \backslash \{i\}, \\
& y_{i',j} \in \mathbb{Z}_{\geq 0}, \ \text{for all } i' \in [I], j \in [J_{i'}].
\end{aligned}
$$

Here, $\boldsymbol{y}$ resumes its meaning as general integer design variables. Solving F-SDOP and $\varepsilon$-$\delta$-KPSDOP gives a front of candidate design solutions for BO-KPSDMP. SP2 of APP then produces the dynamic maintenance Pareto fronts of each of these designs, gathers them all together, and eliminates the dominated solutions to get an approximate

Pareto front. For our purposes, APP could terminate as soon as a solution with negative reduced cost is found. Notably, if a design-only solution has negative reduced cost, then we can skip the SP2 phase, as this is more computationally intensive. We can also opt to find each fixed-design Pareto front for SP2 in ascending order of LFR, starting with the most reliable design as this can roughly be seen as having the most flexibility in terms of dynamic maintenance. Again, if by doing this we find a column with negative reduced cost, we could terminate APP early. Algorithm 4 describes this APP-based method for solving KPSDMP as a step-by-step algorithm.

**Algorithm 4** Approximate KPSDMP Solver

1. Input: Problem parameters $\mathcal{P}$, subsystem index $i$, dual variables $\boldsymbol{\sigma}, \boldsymbol{\beta}$.

2. SP1: Initialise design set $\mathcal{X}_i^* = \emptyset$. Solve F-SDOP, add this solution to $\mathcal{X}_i^*$, record the objective value as $\varepsilon_{\min}$. Set $\varepsilon = \infty$. Repeat until $\varepsilon < \varepsilon_{\min}$:

   - Solve $\varepsilon$-$\delta$-KPSDOP using a MILP solver, obtaining $\boldsymbol{x}_{i,\varepsilon}^*$. Add this solution to $\mathcal{X}_i^*$.
   - Set $\varepsilon = \text{Dec}(\sum_{j=1}^{J_i} \sum_{k=1}^{K_{i,j}^{\max}} \ln(q_{i,j})[\boldsymbol{x}_{i,\varepsilon}^*]_{j,k})$.

   If KPSDMP objective value amongst the design-only solutions of $\mathcal{X}_i^*$ is negative, return the design with the lowest objective value along with its fully-active policy, $(\boldsymbol{x}_i^*, \mu_i^{FA}(\boldsymbol{x}_i^*))$. Otherwise, continue.

3. SP2:

   - Initialise design-policy pair set $\mathcal{XM}_i^* = \emptyset$.
   - For each $\boldsymbol{x}_i \in \mathcal{X}_i^*$, solve BO-DMP for that design, and for each Pareto-optimal policy (except the never repair policy) $\mu_i$ add $(\boldsymbol{x}_i, \mu_i)$ to $\mathcal{XM}_i^*$.

4. Return $(\boldsymbol{x}_i, \mu_i) \in \mathcal{XM}_i^*$ with minimum KPSDMP value.

Algorithm 4 is used to approximately solve the pricing subproblem for CR-SPIDMP-MP-P, for which the overall solution algorithm is given by Algorithm 5.

**Algorithm 5** Approximate CR-SPIDMP-MP-P Solver

1. Input: Problem parameters $\mathcal{P}$, failure penalty $P$, starting columns $\left\{\overline{\mathcal{XM}_i}\right\}_{i\in[I]}$.
2. Initialise: Solve CR-SPIDMP-RMP-P with starting columns, obtaining dual variables $\boldsymbol{\sigma}, \boldsymbol{\beta}$.
3. Set $rc_i = -\infty$ for each $i$. Repeat until $rc_i \geq 0$ for all $i$:
   - For each $i$, solve KPSDMP for the current dual values to obtain entering columns, recording each $rc_i$.
   - Add each new column with $rc_i < 0$ to their respective $\overline{\mathcal{XM}_i}$, resolve CR-SPIDMP-RMP-P, update dual variables $\boldsymbol{\sigma}, \boldsymbol{\beta}$.
4. Return $\left\{\overline{\mathcal{XM}_i}\right\}_{i\in[I]}$.

Algorithm 5 is used to solve mixed-objective versions of the continuous relaxation of the master problem to obtain column sets, and this in turn is used to solve each node of the dichotomic search over the full bi-objective continuous relaxation CR-SPIDMP-MP in Algorithm 6.

**Algorithm 6** Approximate CR-SPIDMP-MP Solver

1. Input: Problem parameters $\mathcal{P}$.
2. Initialise by solving the single objective problems. Solve CR-SPIDMP-MP-F, keeping the column sets $\{\overline{\mathcal{XM}_i}\}_{i\in[I]}$. The cost-only single-objective version of CR-SPIDMP-MP always evaluates to 0 (simply never repair for any given design). Initialise node queue $Q = \{(0, 1)\}$. Set $n \leftarrow 2$, the current number of solutions in the Pareto front
3. Repeat until $Q = \emptyset$:
   - Pop a node off the queue, call this $(r_1, r_2)$
   - Calculate objective weights $w_1, w_2$ for this node following dichotomic method, set $P = w_2/w_1$.
   - Solve CR-SPIDMP-MP-P using Algorithm 5 with starting columns $\{\overline{\mathcal{XM}_i}\}_{i\in[I]}$, and update these sets with all columns found during the procedure.
   - If this solution improves upon the objective for weights $w_1, w_2$ compared to solutions $r_1$ and $r_2$, set $n \leftarrow n + 1$ add two new nodes $(r_1, n), (n, r_2)$ to $Q$.
4. Return column sets $\{\overline{\mathcal{XM}_i}\}_{i\in[I]}$.

Solving CR-SPIDMP-MP in this way generates a large set of feasible columns which we then use to approximately solve SPIDMP-MP via Algorithm 7.

**Algorithm 7** Approximate SPIDMP-MP Solver

1. Input: Problem parameters $\mathcal{P}$.
2. Initialise solution set $X\mathcal{M}^* = \emptyset$.
3. Solve CR-SPIDMP-MP using Algorithm 6, record all the generated columns $\{\overline{X\mathcal{M}_i}\}_{i\in[I]}$.
4. Solve SPIDMP-RMP-F using these columns to get a heuristic solution to the failure minimisation problem. Record this solution in $\mathcal{X}\mathcal{M}^*$.
5. Solve SPIDMP-RMP via the $\varepsilon$-constraint on the failure objective, first solving without the constraint to get an initial value $\varepsilon_{\max}$, then adjusting $\varepsilon$ on each iteration to cut off the previous solution and reoptimise, then add this new solution to $\mathcal{X}\mathcal{M}^*$. Repeat until it reaches the minimum value it can take as determined in step 3.
6. Return $\mathcal{X}\mathcal{M}^*$.

We now summarise precisely what all these different methods are used for. We are solving CR-SPIDMP-MP via column generation i.e., by repeatedly solving CR-SPIDMP-RMP and adding new columns to the sets $\overline{\mathcal{X}\mathcal{M}_i}$, where the column generation methods are constrained to only produce integer-feasible columns. We begin by solving the CR-SPIDMP-MP-F, which we can do almost exactly (up to sensitivity on updating $\varepsilon$) by repeatedly generating columns with minimum negative reduced cost, which we do at each iteration of the column generation procedure (for each subsystem $i$) by solving the pricing subproblem KPSDP via a surrogate problem $\varepsilon$-KCSDP with the current optimal dual variables in the CR-SPIDMP-RMP-F. We do this until we can no longer find a new column with negative reduced cost, at which point the continuous relaxation must be optimal. We then proceed to solve the bi-objective CR-SPIDMP-MP via a

dichotomic method, solving the CR-SPIDMP-MP-P at each node of the search. When solving a node, all columns generated so far can be brought over from previous nodes (including the failure-only problem) to allow for a warm start. Columns are generated here by approximately solving the pricing subproblem KPSDMP, a non-linear MDP Design problem whose optimal solution belongs to the Pareto front of BO-KPSDMP, a linear bi-objective MDP Design problem, which in turn we approximately solve by using APP. Within this method, SP1 of APP is solved by the $\varepsilon$-constraint method along with probability chains for linearisation, giving $\varepsilon$-$\delta$-KPSDOP, which can be solved by any MILP solver. Performing SP2 with the SP1 designs then gives an approximate Pareto front for BO-KPSDMP, and we select the solution with the minimum reduced cost as the approximate solution to KPSDMP. This is then added as a column, and this process continues until no new columns with reduced cost can be found, at which point this node of the dichotomic method is approximately solved. All in all, this approximate column generation procedure allows us to approximately solve the CR-SPIDMP-MP-P for any $P$ as chosen by the dichotomic method. When the dichotomic method terminates, and an approximate Pareto front for the CR-SPIDMP-RMP has therefore been produced, all columns generated for each subsystem $i$ can be included in $\overline{\mathcal{XM}_i}$, and these sets can be used to solve SPIDMP-RMP to integer optimality, providing an integer-feasible approximate Pareto front for SPIDMP-MP.

Algorithm 7 provides a heuristic algorithm for obtaining an approximate Pareto front of solutions for SPIDMP-MP, a problem which is exactly equivalent to DC-SPIDMP, which in turn is an overly constrained (separable policies only) version of our base problem BO-SPIDMP. However, the potential for the application of well known methods from the literature can still be investigated to obtain better solutions. Section 6.3.6 now follows with a discussion of two such methods: cutting planes and branch-and-price.

| Acronym | Full Meaning | Purpose |
|---|---|---|
| BO-SPIDMP | Bi-Objective Series-Parallel Design and Maintenance Problem | Main problem (sometimes just referred to as SPIDMP) |
| DC-SPIDMP | Decomposed Coherent SPIDMP | Approximate block-diagonal version of SPIDMP |
| CR- | Continuous Relaxation | Variant of a problem that allows for pricing subproblems and column generation |
| BO- | Bi-Objective | Indicates that a problem has two objectives |
| -(R)MP | (Reduced) Master Problem | Dantzig-Wolfe Decomposition of DC-SPIDMP with some (reduced) or all columns |
| -P | Penalty | Mixed-Objective version of a problem with second objective weight $P$ |
| -F | Failure-Only | Variant of a bi-objective problem which only optimises for the second (failure probability) objective |
| SPIDMP-(R)MP | SPIDMP (Reduced) Master Problem | (R)MP of DC-SPIDMP |
| SPIDMP-(R)MP-F | SPIDMP-(R)MP with failure objective only | Single-objective variant of the above |
| CR-SPIDMP-(R)MP-F | Continuous relaxation of the above | Single-objective problem variant which can be solved via column generation |
| KPSDP | Knapsack Penalised Subsystem Design Problem | Nonlinear pricing subproblem for the above |
| BO-KPSDP | Bi-Objective KPSDP | Bi-objective MILP whose Pareto front contains an optimal solution to the KPSDP |
| $\varepsilon$-KCSDP | $\varepsilon$-Knapsack-Constrained Subsystem Design Problem | $\varepsilon$-constrained version of BO-KPSDP |
| CR-SPIDMP-(R)MP-P | Continuous relaxation of SPIDMP-(R)MP with mixed objectives | Mixed-objective problem variant (with second objective weight $P$) which can be solved via column generation |
| KPSDMP | Knapsack-Penalised Subsystem Design and Maintenance Problem | Non-linear pricing subproblem for the above |
| BO-KPSDMP | Bi-objective KPSDMP | Bi-objective MDP Design MILP whose Pareto front contains an optimal solution to KPSDMP |
| $\varepsilon$-$\delta$-KPSDOP | $\varepsilon$-constrained $\delta$-augmented Knapsack Penalised Subsystem Design-Only Problem | Generates candidate designs for BO-KPSDMP under the APP methodology |
| F-SDOP | Failure-Only Subsystem Design-Only Problem | Generates most reliable subsystem design for BO-KPSDMP |

Table 6.3.1: Acronyms for problem variants and pricing subproblems

### 6.3.6 Extensions

#### Cutting Planes

The knapsack constraints $C'\boldsymbol{x} \leq \boldsymbol{d}$ in the original problem BO-SPIDMP permit a set of *valid inequalities* $\mathcal{VI}$, which is a set of inequalities which are satisfied for all integer-feasible solutions to the knapsack constraints, but which are violated by at least one solution to its continuous relaxation. Constraints of this type, often called *cuts* or *cutting planes*, can be added to BO-SPIDMP to strengthen its linear relaxation, and can in turn be translated into equivalent constraints in terms of the $\lambda$ variables in SPIDMP-RMP. Each extra valid inequality adds another dual variable to be included in any pricing subproblems alongside the original $\boldsymbol{\sigma}$ and $\boldsymbol{\beta}$ dual variables. Let $U^i\boldsymbol{x}_i \leq \boldsymbol{v}^i$ denote a set of valid inequalities for block $i$ in terms of the original variables; these get translated into the RMP variables as $U^{\Lambda_i}\boldsymbol{\lambda}_i \leq \boldsymbol{v}^i$ with dual variables $\boldsymbol{\theta}$, and tighten the linear relaxation. For solving CR-SPIDMP-RMP-F, we add $-\boldsymbol{\theta}^T U^i\boldsymbol{x}_i$ to the objective function of the pricing subproblem KPSDP, which via the bi-objective linearisation ends up being included in the $\varepsilon$-constraint of $\varepsilon$-KCSDP. This new pricing subproblem helps to find promising columns given that some solutions that were linear combinations of the $\boldsymbol{\lambda}_i$ have been "cut off" by the valid inequalities. For CR-SPIDMP-RMP-F, the same change occurs for the objective function of the pricing subproblem KPSDMP, which again is linearised by being turned into the bi-objective problem BO-KPSDMP, with the new dual terms being included in the first objective. This naturally extends to being included in the objective function of $\varepsilon$-$\delta$-KPSDOP alongside the $\boldsymbol{\sigma}$ terms, which is used to solve the first stage of APP. The dual terms of course have no bearing on SP2. As such, the inclusion of these cutting planes has only a very minor impact on the pricing subproblems and their respective solution methodologies.

Of course, not all valid inequalities are included at the start. Instead, whenever a linear relaxation CR-SPIDMP-RMP-F/P is solved and the new (near-)optimal solution is still fractional in the original variables, a subproblem called the *separation problem* is

solved to generate a cut that cuts off this solution whilst leaving all integral solutions in tact. This itself is an NP-hard problem, but a reasonably small one for which many exact and heuristic methods exist for the knapsack constraints that we consider (Kaparis and Letchford, 2010). Usually, a cutting planes approach like this would eventually (perhaps after a very large number of cuts) lead to the optimal integral solution being found. However, as we rely on heuristic pricing, this isn't necessarily true for us. A cutting planes algorithm would however get us to a point where we do have a good integral solution, in the sense that our heuristic pricing cannot find any improving columns for any subsystem. Additionally, early termination of the cutting planes method would at least allow us to close the MILP gap, and generate more columns to be considered when solving SPIDMP-RMP. We now move on to consider the *Branch-and-Price* method.

### Single-Objective Branch and Price

A state-of-the-art exact solution methodology for single-objective MILPs with a block diagonal structure is *branch-and-price*, a methodology by which the original problem (i.e., the problem before it has been transformed to a (reduced) master problem via Dantzig-Wolfe decomposition) is being solved by branch-and-bound, and each node (i.e., continuous relaxation paired with some branching decisions) of the branch and bound tree is being solved exactly by column generation. We discuss the applicability of branch-and-price to DC-SPIDMP, and our discussion will yield a direction of future research for the branch-and-price methodology in general.

Firstly, the standard branch-and-price algorithm only applies to single-objective problems, so we must choose a scalarisation of DC-SPIDMP. Note that this immediately raises a problem of computational difficulty – a full branch-and-price tree will have to be solved for a great number of different scalarisations; we will return to this point later. Using the mixed-objective method would yield the non-trivial problem of determining the penalty $P$ to be used when solving CR-SPIDMP-MP-P, as this formulation

is obtained by transforming the second objective from DC-SPIDMP. The $\varepsilon-$constraint method, applied to the second objective, is a more appropriate scalarisation as this can more readily be dealt with.

The scalarisation can then be solved via branch-and-bound. To be precise, whenever we refer to "branching" throughout this passage, we always refer to branching on the original $x$ design variables, *not* on the disaggregated $\lambda$ variables. Note that while branch and bound is most often associated with integer linear programs, it is of course applicable to non-linear integer programs like DC-SPIDMP; the notions of solving continuous relaxations and adding branching decisions still apply trivially. After applying the appropriate transformation to this $\varepsilon$-constraint, Dantzig-Wolfe decomposition can be applied at any node of the branch and bound tree, yielding a version of CR-SPIDMP-MP with the second objective as a linear $\varepsilon$-constraint (call this $\varepsilon$-CR-SPIDMP-MP), and as such this problem can be solved via column generation. The pricing subproblem in this case is in fact identical to KPSDMP, the pricing subproblem for CR-SPIDMP-MP-P, except now the penalty parameter $P$ is the dual variable of the $\varepsilon$-constraint for failure, rather than a fixed objective weighting. In turn, this pricing subproblem is solved approximately via the APP methodology. This demonstrates a shortfall, in that our application of branch-and-price here is still only a heuristic, and we do not solve the pricing subproblem exactly. This could lead to a case where we fail to find a column with reduced cost, even though one exists, which in turn could lead to the premature fathoming of nodes from the search tree. Nevertheless, the use of branch-and-price still represents an improvement on simply solving the root node.

We should also consider the difficulties added by the branching decisions, which span a few points:

- The problem of deciding which variable to branch on, which is a general research question in the branch and bound literature, alongside the determination of the search strategy (i.e., choosing which node in the queue to try next). Both of these

questions are well known in the literature (Morrison et al., 2016), and different strategies could be tried via numerical experimentation.

- The branching decisions of a node could be integer-infeasible. This should be checked via a small MILP to ensure that the branching decisions permit an integer solution that satisfies the knapsack constraints and allows at least one component per subsystem (we ignore the $\varepsilon$-constraint for now). A node that fails this test should be immediately fathomed.

- All columns generated so far that violate the branching decisions must be ignored in this node. It may be the case that the remaining columns lead to an infeasible $\varepsilon$-CR-SPIDMP-RMP, making it impossible to generate columns via the pricing subproblems. An initial set of feasible columns that satisfy the branching decisions can be identified as follows:

  1. take the design used to prove that this node is integer-feasible, and use it as a starting feasible solution for CR-SPIDMP-MP-F (the continuous failure minimisation problem),
  2. proceed to start solving CR-SPIDMP-MP-F via column generation, and terminate early when the objective function is such that the $\varepsilon-$constraint is satisfied (if the constraint cannot be satisfied, fathom the node);
  3. add these columns to this node of the tree.

- If the node has not been fathomed and has a feasible RMP solution, we can proceed to solve via column generation, i.e., solving KPSDMP using APP. Of course, any pricing subproblem is updated to directly include the branching decisions, so it continues to only generate feasible columns.

This concludes our discussion of how a branch-and-price method could be used to solve $\varepsilon$-constraint scalarisations of DC-SPIDMP. Before we move on, it is of course

worth mentioning that the cutting planes we have described previously can be built into the branch-and-price tree to improve the linear relaxation at any given node. This allows us to do "more work" on each node, which tends to reduce the number of nodes that have to be processed. This method can also be used as a column generating heuristic: the branch-and-price procedure could easily be given a time limit and, if it doesn't terminate of its own accord, all columns generated can be used in an RMP to obtain heuristic solutions.

As noted previously, all of this suffers from the fact that a full branch-and-price tree must be solved for each value of $\varepsilon$ for which we are optimising. The following section looks towards a modern methodology that could alleviate this issue.

### Bi-Objective Branch and Price

Parragh and Tricoire (2019) introduced an algorithm called *bi-objective branch and bound*. It solves bi-objective *integer* linear programs (*not* MILPs) to Pareto-optimality (including unsupported solutions) via branch and bound by solving each node of the tree to Pareto-optimality. Nodes are only fathomed if their whole Pareto front is weakly dominated with respect to the incumbent integer-feasible Pareto front. They additionally introduced the notion of *"branching on objective space"*, whereby a node whose Pareto front is only dominated by the incumbent integer Pareto front in part, but not in whole, is split into multiple subproblems: one for each connected piece of the non-dominated front. For example, suppose for some node of the search tree, the node's relaxation solutions were worse than some incumbent integer solutions for objective weights $w \in [0.4, 0.6]$ for mixed-objective scalarisation $f_w(x) = w \cdot f_1(x) + (1-w) f_2(x)$, but that the relaxation solutions were better than incumbent integer solutions for other weights. If we were to branch on this node in terms of the decision variables, we would also branch on the objective space producing some subproblems that only solve for weights $w \in [0, 0.4]$ and others that only solve for $w \in [0.6, 1]$. This allows the process

to avoid unnecessarily optimising a linear relaxation for values of $w$ which we know it won't work well for, and also allows the procedure to make different branching decisions for different parts of the objective space. In the same work, the authors additionally introduced *bi-objective branch and price*, in which the linear relaxations of each node are solved via column generation. Of course, this remained limited to pure integer variables. More recently, Parragh et al. (2022) extended this to a bi-objective branch-and-Benders' cut algorithm, tackling the other two-stage type of block-diagonal integer linear program.

Adelgren and Gupte (2022) extended this to a bi-objective branch-and-bound algorithm for MILPs. The authors also pay more attention to other aspects of the branch-and-bound algorithm along the lines of state-of-the-art MILP solvers, considering aspects such as presolve, preprocessing, probing, and measurements of the gap between the LP relaxations and the incumbent Pareto front. However, they do not extend their algorithm to the branch-and-price context. As such, there is a small gap in the literature for a bi-objective branch-and-price algorithm for mixed-integer variables, which would extend the bi-objective branch and bound algorithm of Adelgren and Gupte (2022) to allow for column generation at each node. The development of such a generic methodology should of course be first tested on well-known problem instances which can be solved exactly (e.g. those with known exact methods for the pricing subproblem, rather than the heuristics that we rely on for our problem). A worthwhile consideration for a bi-objective branch-and-price solver, and one to our knowledge not otherwise mentioned in the literature, is that a global bi-objective RMP with all (or some) of the columns generated so far can always be maintained and repeatedly re-optimised throughout the tree search in order to keep a strong upper bound set (e.g. approximate integer-feasible Pareto front), allowing for far more aggressive pruning of the search tree. This of course comes with the implication of computational time spent on re-solving the RMP; computational experiments would be required to determine a good

frequency of re-optimisation.

Within a bi-objective branch-and-price framework, DC-SPIDMP could be solved approximately by heuristically solving CR-SPIDMP-MP (with branching decisions and possibly cuts) at each node of the tree using Algorithm 6, accounting for the branching decisions along similar lines to those mentioned in our discussion of single-objective branch-and-price (namely, checking the integer-feasibility of this node, and using such a feasible design as a starting point for column generation if the present columns do not provide a feasible starting solution). Ultimately, even if all of this was implemented, it would still only be a heuristic for DC-SPIDMP due to the reliance on heuristic pricing, and in turn it is still only a heuristic for BO-SPIDMP due to the limitation of separable policies. This concludes our discussion of the many avenues through which we can attempt to solve BO-SPIDMP. Section 6.4 now follows with a computational study, in which we explore the ability of a standard MILP solver to solve problem instances, and explore the implementation of the no-mixing heuristic.

## 6.4 Computational Study

### 6.4.1 Problem Set

We will construct problem sets based on the 14-subsystem problem from Fyffe et al. (1968), as laid out in Table 6.4.1. This is a problem instance with 14 subsystems, each of which has 3 or 4 different component types, and where each component has a reliability value $p$, installation cost $C_1$, and weight $C_2$, where the latter two contribute to budget and weight constraints, respectively. We can construct smaller problem instances by considering subsets of the subsystems and by considering subsets of each subsystem's components. This original problem set does not consider the following parameters in our model: $r, u, \tau, \alpha$, which we recall are the repair cost rate, usage cost rate, repair rate, and failure rate, respectively. As a baseline, we set $r_{i,j} = 100$, $u_{i,j} = 1$, and $\tau_{i,j} = 1$

| Subsystem | Component 1 | | | Component 2 | | | Component 3 | | | Component 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $p$ | $C_1$ | $C_2$ | $p$ | $C_1$ | $C_2$ | $p$ | $C_1$ | $C_2$ | $p$ | $C_1$ | $C_2$ |
| 1 | 0.90 | 1 | 3 | 0.93 | 1 | 4 | 0.91 | 2 | 2 | 0.95 | 2 | 5 |
| 2 | 0.95 | 2 | 8 | 0.94 | 1 | 10 | 0.93 | 1 | 9 | - | - | - |
| 3 | 0.85 | 2 | 7 | 0.90 | 3 | 5 | 0.87 | 1 | 6 | 0.92 | 4 | 4 |
| 4 | 0.93 | 3 | 5 | 0.87 | 4 | 6 | 0.85 | 5 | 4 | - | - | - |
| 5 | 0.94 | 2 | 4 | 0.93 | 2 | 3 | 0.95 | 3 | 5 | - | - | - |
| 6 | 0.99 | 3 | 5 | 0.98 | 3 | 4 | 0.97 | 2 | 5 | 0.96 | 2 | 4 |
| 7 | 0.91 | 4 | 7 | 0.92 | 4 | 8 | 0.94 | 5 | 9 | - | - | - |
| 8 | 0.81 | 3 | 4 | 0.90 | 5 | 7 | 0.91 | 6 | 6 | - | - | - |
| 9 | 0.97 | 2 | 8 | 0.99 | 3 | 9 | 0.96 | 4 | 7 | 0.91 | 3 | 8 |
| 10 | 0.83 | 4 | 6 | 0.85 | 4 | 5 | 0.90 | 5 | 6 | - | - | - |
| 11 | 0.94 | 3 | 5 | 0.95 | 4 | 6 | 0.96 | 5 | 6 | - | - | - |
| 12 | 0.79 | 2 | 4 | 0.82 | 3 | 5 | 0.85 | 4 | 6 | 0.90 | 5 | 7 |
| 13 | 0.98 | 2 | 5 | 0.99 | 3 | 5 | 0.97 | 2 | 6 | - | - | - |
| 14 | 0.90 | 4 | 6 | 0.92 | 4 | 7 | 0.95 | 5 | 6 | 0.99 | 6 | 9 |

Table 6.4.1: 14-subsystem problem of Fyffe et al. (1968).

for all components $(i, j)$. We then set $\alpha_{i,j}$ by solving $p_{i,j} = \tau_{i,j}/(\tau_{i,j} + \alpha_{i,j})$ for each $i, j$.

### 6.4.2 Scalability of the MILP Solver

To begin, we investigate the extent to which problem instances can be solved exactly by a MILP solver (namely, Gurobi 11 in Julia 1.9.3). We first consider BO-SPIDMP without the first objective, so that we are only optimising over the failure minimisation objective, and we call this SPIDMP-F. This is in fact quite an inefficient formulation of a series-parallel RAP (there are far more decision variables than there are feasible design solutions, and the MDP action choices will always be "repair everything"); however, it is worth investigating how well a MILP solver can return single optimal solutions to this problem before we try to obtain Pareto fronts with many solutions for the bi-objective problem. We start by considering problem instances of SPIDMP-F derived from Table 6.4.1 which only have one component type per subsystem (namely type 1). We consider problems with systems made up of the first $i$ subsystems from our 14 subsystem problem for different values of $i$, and consider a single simple knapsack con-

straint which limits the total number of components to $d$ (so each knapsack coefficient is 1, and the cost and weight are ignored for these instances). We call these *cardinality* constraints. The results of testing these problem types are given in Table 6.4.2. For each number of subsystems, we omit most of the values of $d$ for which the problem is solved easily, as these are not of interest.

Table 6.4.2 presents our results. For the instances with one subsystem, we were able to solve instances with cardinality constraints up to 11 with ease. For cardinalities of 12 and 13, we see the operational cost go up and then down again, indicating numerical error. The solver at least continued to return non-zero values for failure rate up to a cardinality of 22, whereas a cardinality constraint of 23 led to an optimal system which was too reliable to be properly captured within the tolerances of the MILP solver, even where the solver parameters have been chosen to ensure maximum numerical stability. While not stated in the table, we also noted that changing solver parameter settings often greatly affected solutions above a cardinality of 12.

For two subsystem problems, we could solve problems up to a cardinality of 14 with ease. A cardinality constraint of 15 led to some instability during the solving process, and a cardinality constraint of 16 led to a solution that violated certain tolerances.

For systems with 3, 4 or 5 subsystems, it appears that solution times increase by orders of magnitude with the cardinality constraint, with some numerical instability. It appears that we cannot hope to solve 4-subsystem instances with constraints that allow for anything over 2 layers of redundancy per subsystem, and for 5 subsystems we cannot manage even that. For systems with more than 6 subsystems, we quickly begin to run into memory issues, even for cardinality constraints that allow for little more than 1 component per subsystem. For 10 subsystems, we have no hope of solving even a problem that only allows for 1 component per subsystem.

Altogether, these first results significantly narrow our scope for solving the full bi-objective problem, even for problems with only one component type per subsystem. It

appears that problem instances with: 1) a good amount of redundancy (i.e., allowing for at least 2 components per subsystem) and 2) reasonable solve times (important given many solutions can be in a Pareto front), are limited to systems with up to 3 subsystems. This roughly coincides with our results from Section 5.4.3, where solution times for the maintenance problems for four-subsystem one-type problems seemed to be the limit; it's reasonable to expect that adding the design aspect places these problems out of reach.

Table 6.4.3 gives the results of a similar experiment, but allowing for the first two component types in each subsystem. These results are far more dire in terms of the implications for solving interesting problem instances exactly. For single subsystem problems, cardinalities up to and including 8 solve without issue, however a cardinality of 8 leads to numerical issues which aren't detected by the solver, but which are known to be false. For two subsystem problems, cardinalities up to and including 6 solve without issue, a cardinality of 7 solves in 870 seconds (with the caveat that for this instance we use Gurobi's "no relaxation" heuristic for 160 seconds to tighten the upper bound), and a cardinality of 9 leads to numerical difficulties and significant slowdown in the root relaxation. For three subsystem problems, cardinalities up to 5 solve easily. A cardinality of 6 is interesting in that it obtains a MILP gap of 0.27% in 103 seconds, but requires 705 seconds to solve fully. A cardinality of 7 cannot be solved in reasonable time. For four subsystems and a cardinality of 5, we encounter similar behaviour to the cardinality of 6 for the three subsystem problem, in that a good bound is obtained quickly but the full solution takes far longer. This similarly seems to be true for a cardinality of 6, but this does not solve in reasonable time. For five subsystems, we can only solve with a cardinality of 5, and cannot even solve the root relaxation for a cardinality of 6. Six subsystems with cardinality 6 encounters numerical issues, and seven subsystems with a cardinality of 7 runs out of system memory. These results demonstrate that we are extremely limited in our ability to solve this time of problem

exactly. On one hand, this very much motivates the need for the heuristic approaches that we have described; however, it also severely limits our ability to compare the performance of our heuristics against exact solutions.

### 6.4.3 No Mixing Heuristic on Small Instances

We wish to compare the performance of our No-Mixing heuristic (Algorithm 1) against the exact solver. We can expect this to perform reasonably well for problems with only one component type per subsystem, as the only approximation made by the heuristic in this case is the imposition of a coherent separable policy, and these were demonstrated as performing well in Chapter 5 (Section 5.4.3). We construct problem instances by first forming systems by taking contiguous subsets of the original 14 subsystems, considering only their first component type, and then attaching a cardinality constraint. For example, the problem with subsystems 2:4 with cardinality 8 is a problem that considers a system made up of subsystems 2, 3 and 4, with an overall limit of 8 components across the whole system, where each subsystem only allows for the first component type as set out in Table 6.4.1. For the sake of comparison, we only do this for 2- and 3-subsystem problems, as problems with four subsystems or more are extremely limited in terms of the cardinality values for which they are solvable in reasonable time, and these low cardinality values make for quite uninteresting problem instances. Table 6.4.4 and 6.4.5 show the results for 2-subsystem problems, and Table 6.4.6 and 6.4.7 show the results for 3-subsystem problems.

We first consider the 2-subsystem problems. One detail we must specify is how we update the $\varepsilon$-constraint in the No-Mixing Heuristic. For these problems, we always update $\varepsilon$ to be $0.99 \cdot LHS$, where $LHS$ is the optimal value of the left-hand-side of the $\varepsilon$ constraint from the previous solution – this ensures that the previous solution is "cut off". Across Tables 6.4.4 and 6.4.5, we see similar performance to the decomposition heuristic of Chapter 5 and demonstrated in Section 5.4.3: the heuristic runs in a fraction

| Number of Subsystems | $d$ | Notes |
|---|---|---|
| 1 | 11 | Solves in 0.31 seconds, OpCosts = 97.6, LFR = -23.6 |
| | 12 | Solves in 0.32 seconds, OpCosts = 101.5, LFR = -24.75 |
| | 13 | Solves in 0.31 seconds, OpCosts = 98.2, LFR = -25.02. |
| | 23 | Solves in 1.3 seconds, LFR -32.7 |
| | 24 | Solves in just over a second, returns a failure rate of 0 due to numerical error |
| 2 | 14 | Solves in 298 seconds, LFR = -17.5 |
| | 15 | Variables dropped from basis during barrier method of root relaxation. Forcing dual simplex at nodes instead solves in 496 seconds, LFR = -18.5. |
| | 16 | Receive warnings that "max constraint violation" and "max bound violation" exceeds tolerance. Solver terminates in 6400 seconds with LFR -20.2. |
| 3 | 9 | Solves in 170 seconds with LFR = -5.5 |
| | 10 | Solves in 1856 seconds with LFR = -6.4 |
| | 11 | Solves in 12740 seconds with LFR = -7.2. |
| 4 | 7 | Solves in 71.5 seconds with LFR = -2.5. |
| | 8 | Solves in 397 seconds with LFR = 3.2. |
| | 9 | Takes around 600s alone to solve the root relaxation, then made no further progress within an hour. No solution in reasonable time. |
| 5 | 6 | Solves in 71.9 seconds with LFR = -1.3 |
| | 7 | Solves in 3486 seconds with LFR = -1.6. |
| | 8 | Takes over 5000 seconds to solve root relaxation alone. Some numerical issues. No solution in reasonable time. |
| 6 | 7 | Solves in 2694 seconds, LFR = -1.28. |
| | 8 | Barely fits into memory (16GB almost full). 5722s to solve root relaxation alone. No solution in reasonable time. |
| 7 | 7 | Solves in 12 seconds, LFR = -0.85. |
| | 8 | Model doesn't fit into memory (started using swap file whilst building model) |
| 8 | 8 | Solves in 67.6 seconds, LFR = -0.62. |
| | 9 | Model doesn't fit in memory. |
| 9 | 9 | Solves in 715 seconds, LFR = -0.597. |
| | 10 | Model doesn't fit in memory. |
| 10 | 10 | Model doesn't fit in memory. |

Table 6.4.2: Result of solving failure-only SPIDMP using a MILP solver for problem instances with different numbers of subsystems, cardinality constraints, and one component type per subsystem.

| Number of Subsystems | $d$ | Notes |
|---|---|---|
| 1 | 8 | Solves in 1.22 seconds with OpCost $\approx$ 57.00 and LFR = -21.274. |
| | 9 | Solves in 2.9 seconds but returns an OpCost of 58.04. We know that OpCost must increase by 7 for each extra component (most reliable of the two components has reliability 0.93, so repair costs are 0.07*100 for each extra component). This therefore suggests numerical error. |
| 2 | 6 | Solves in 10.3 seconds with LFR = -7.67. |
| | 7 | Solves in 870 seconds* with LFR = -8.8. |
| | 8 | Root relaxation encounters numerical difficulties. |
| 3 | 5 | Solves in 37.5 seconds with LFR = -2.75. |
| | 6 | Obtains a MIP Gap of 0.27% in 103 seconds. Fully solves in 705 seconds with LFR = -4.06 |
| | 7 | Root relaxation takes over an hour. Solution not returned in reasonable time. |
| 4 | 5 | Obtains MIP Gap of 2.41% in 26 seconds. Solves fully in 331.5 seconds with LFR = -1.68. |
| | 6 | 465615 state-action pairs. Reaches an 8.33% optimality gap after 1700s. Left running overnight from 5pm to 10am, branch-and-cut log indicated no further progress. |
| 5 | 5 | Solves in 4.7 seconds with LFR = -1.19. |
| | 6 | Around 800000 state-action pairs. Extremely slow progress when examining the barrier log for root relaxation, eventually causing a crash. |
| 6 | 6 | Solver terminates in 144.5 seconds reporting an LFR of -1.17, however this is with max constraint and max bound violations. |
| 7 | 7 | Runs out of system memory. |

Table 6.4.3: Result of solving failure-only SPIDMP using a MILP: solver for problem instances with different numbers of subsystems and cardinality constraints, and two component types per subsystem.

of the time needed by the exact solver, but produces a Pareto front with fewer solutions, where some of these solutions are in fact Pareto-dominated by the exact solutions. Comparing the number of solutions found by the heuristic solver to the number of those solutions which are in fact dominated by an exact solution, we can see that it is often the case that a large proportion of the solutions found are Pareto-suboptimal. However, the generally low values for Mean REPOG suggest that the extent of this suboptimality is minor. We also see an interesting pattern forming in the Max REPOG values, in that the same values often "carry forward" for the same subsystems as cardinality increases. This suggests that as cardinality increases, previous solutions from both the exact and heuristic methods often remain Pareto-optimal, but still demonstrate the same optimality gap for the heuristic solution. The fact that this value doesn't increase with cardinality perhaps suggests that the heuristic is often weakest for low cardinality systems, and for the less reliable parts of the Pareto front. In terms of the values taken by Max REPOG, we see that they mostly fall below 10%, aside from one problem instance (4:5 with cardinality 5) which gets a Max REPOG value of 14.37%. Luckily, the other aspects of the heuristic soluion for this problem are more encouraging, with a Mean REPOG of 5.7% (far lower than the Max REPOG), only 4 out of 20 heuristic solutions being Pareto-suboptimal, and an approximate Pareto front with 20 solutions compared to the exact front's 30 solutions. We also see that this gap does not carry forward as cardinality increases.

Moving on to 3-subsystem problems, first note that here we use a factor of 0.999 instead of 0.99 to cut off the previous solution when updating the $\varepsilon$ constraint. Tables 6.4.6 and 6.4.7 show a similar story. Again, the heuristic solves in a fraction of the time taken by the exact solver – always in less than 4 seconds, whereas the exact solver takes in excess of 100000 seconds for some of the larger (cardinality 9) instances. Of course, these approximate fronts lack in terms of the number of solutions, and many of these solutions end up being slightly dominated, but the Mean REPOG values again

suggest that the extent of this domination is minor. All of the Mean REPOG values fall below 5% except for one instance, 8:10 with cardinality 5, which had a Mean REPOG of 6.89% over 4 dominated solutions out of 14. This instance also suffered from a high Max REPOG of 15.62%. This lends further credibility to the idea that the heuristic performs worse at low cardinalities. However, one might argue that dynamic maintenance policies are least desirable for such instances, due to their lower reliability and reduced flexibility (i.e., smaller space of healthy states) in terms of maintenance policy. The other instances with subsystems 8:10 also suffered from high Max REPOG values. In terms of other high Max REPOG values, the remainder of instances with Max REPOG values above 10% all used subsystems 3:5.

| | | Exact | | Heuristic | | | | |
|---|---|---|---|---|---|---|---|---|
| Subsystems | Cardinality | Time | Solutions | Time | Solutions | Dominated Solutions | Max REPOG (%) | Mean REPOG (%) |
| 1:2 | 5 | 6.31 | 41 | 0.1 | 16 | 2 | 5.94 | 3.87 |
| 1:2 | 6 | 20.94 | 57 | 0.21 | 27 | 9 | 5.94 | 2.07 |
| 1:2 | 7 | 76.91 | 76 | 0.43 | 45 | 16 | 5.94 | 2.25 |
| 1:2 | 8 | 216.42 | 98 | 0.68 | 69 | 24 | 5.94 | 2.29 |
| 1:2 | 9 | 1025.56 | 156 | 1.58 | 88 | 39 | 5.94 | 1.81 |
| 1:2 | 10 | 2486.84 | 192 | 3.02 | 121 | 56 | 5.94 | 1.73 |
| 2:3 | 5 | 3.82 | 37 | 0.15 | 20 | 4 | 7.48 | 2.18 |
| 2:3 | 6 | 18.1 | 53 | 0.2 | 32 | 8 | 7.48 | 2.44 |
| 2:3 | 7 | 63.2 | 89 | 0.42 | 46 | 11 | 7.48 | 2.17 |
| 2:3 | 8 | 253.46 | 130 | 0.73 | 65 | 24 | 7.48 | 2.16 |
| 2:3 | 9 | 842.75 | 152 | 1.5 | 94 | 39 | 7.48 | 2.03 |
| 2:3 | 10 | 2187.87 | 184 | 2.39 | 122 | 54 | 7.48 | 1.98 |
| 3:4 | 5 | 5.03 | 35 | 0.15 | 19 | 6 | 6.42 | 2.98 |
| 3:4 | 6 | 21.94 | 65 | 0.26 | 31 | 13 | 6.42 | 2.57 |
| 3:4 | 7 | 104.23 | 110 | 0.44 | 43 | 17 | 6.42 | 2.54 |
| 3:4 | 8 | 251.67 | 105 | 0.89 | 66 | 30 | 6.42 | 2.02 |
| 3:4 | 9 | 709.79 | 129 | 1.38 | 84 | 39 | 6.42 | 1.97 |
| 3:4 | 10 | 2396.53 | 204 | 2.55 | 111 | 54 | 6.42 | 2.11 |
| 4:5 | 5 | 5.71 | 30 | 0.15 | 20 | 4 | 14.37 | 5.7 |
| 4:5 | 6 | 21.57 | 72 | 0.27 | 29 | 9 | 9.45 | 3.23 |
| 4:5 | 7 | 52.62 | 73 | 0.47 | 48 | 13 | 6.79 | 2.49 |
| 4:5 | 8 | 313.58 | 128 | 0.94 | 68 | 18 | 6.79 | 2.27 |
| 4:5 | 9 | 498.04 | 109 | 1.9 | 90 | 23 | 6.79 | 1.91 |
| 4:5 | 10 | 1938.69 | 144 | 2.96 | 109 | 36 | 6.79 | 1.75 |
| 5:6 | 5 | 4.32 | 35 | 0.16 | 17 | 0 | 0.0 | 0.0 |
| 5:6 | 6 | 21.62 | 71 | 0.26 | 30 | 5 | 3.32 | 1.61 |
| 5:6 | 7 | 52.31 | 63 | 0.37 | 42 | 5 | 6.24 | 1.86 |
| 5:6 | 8 | 167.09 | 86 | 0.95 | 66 | 10 | 6.24 | 1.23 |
| 5:6 | 9 | 444.94 | 89 | 1.43 | 95 | 21 | 6.24 | 1.31 |
| 5:6 | 10 | 1038.51 | 99 | 2.56 | 122 | 39 | 6.24 | 1.81 |
| 6:7 | 5 | 4.11 | 34 | 0.2 | 18 | 1 | 6.23 | 6.23 |
| 6:7 | 6 | 16.33 | 55 | 0.26 | 29 | 6 | 6.23 | 2.74 |
| 6:7 | 7 | 58.42 | 65 | 0.4 | 47 | 10 | 5.05 | 2.44 |
| 6:7 | 8 | 149.81 | 83 | 0.76 | 70 | 21 | 6.23 | 2.31 |
| 6:7 | 9 | 474.17 | 111 | 1.39 | 92 | 32 | 6.5 | 2.7 |
| 6:7 | 10 | 1162.1 | 105 | 2.03 | 117 | 44 | 6.53 | 2.25 |
| 7:8 | 5 | 4.59 | 37 | 0.13 | 19 | 7 | 6.53 | 2.9 |
| 7:8 | 6 | 24.13 | 60 | 0.23 | 32 | 14 | 6.53 | 2.55 |
| 7:8 | 7 | 86.1 | 92 | 0.39 | 42 | 17 | 6.53 | 2.59 |
| 7:8 | 8 | 416.71 | 129 | 0.84 | 58 | 28 | 6.53 | 2.41 |
| 7:8 | 9 | 1039.55 | 192 | 1.62 | 79 | 44 | 6.53 | 2.17 |
| 7:8 | 10 | 2300.87 | 242 | 2.45 | 99 | 60 | 6.53 | 2.07 |

Table 6.4.4: Comparison between exact and no-mixing heuristic methods for 2-subsystem problems (Part I)

| | | Exact | | Heuristic | | | | |
|---|---|---|---|---|---|---|---|---|
| Subsystems | Cardinality | Time | Solutions | Time | Solutions | Dominated Solutions | Max REPOG (%) | Mean REPOG (%) |
| 8:9 | 5 | 3.92 | 40 | 0.17 | 20 | 5 | 8.76 | 4.06 |
| 8:9 | 6 | 14.85 | 59 | 0.24 | 36 | 8 | 6.07 | 3.49 |
| 8:9 | 7 | 67.26 | 70 | 0.43 | 56 | 18 | 8.66 | 2.83 |
| 8:9 | 8 | 218.76 | 111 | 0.81 | 69 | 27 | 5.47 | 2.66 |
| 8:9 | 9 | 987.78 | 183 | 1.3 | 89 | 37 | 5.47 | 2.63 |
| 8:9 | 10 | 1820.87 | 176 | 2.12 | 111 | 56 | 6.5 | 2.8 |
| 9:10 | 5 | 3.85 | 36 | 0.1 | 20 | 2 | 7.42 | 3.71 |
| 9:10 | 6 | 17.48 | 60 | 0.28 | 35 | 8 | 9.06 | 3.9 |
| 9:10 | 7 | 69.93 | 83 | 0.43 | 48 | 14 | 9.06 | 2.84 |
| 9:10 | 8 | 227.61 | 109 | 0.77 | 62 | 22 | 9.06 | 2.58 |
| 9:10 | 9 | 911.77 | 177 | 1.21 | 82 | 33 | 9.06 | 2.43 |
| 9:10 | 10 | 1603.55 | 155 | 1.86 | 114 | 54 | 9.06 | 2.41 |
| 10:11 | 5 | 4.63 | 36 | 0.16 | 20 | 7 | 9.04 | 4.17 |
| 10:11 | 6 | 22.67 | 62 | 0.24 | 35 | 11 | 9.04 | 2.69 |
| 10:11 | 7 | 94.97 | 92 | 0.32 | 43 | 13 | 9.04 | 3.55 |
| 10:11 | 8 | 295.22 | 137 | 0.72 | 58 | 26 | 9.04 | 2.78 |
| 10:11 | 9 | 881.69 | 147 | 1.34 | 84 | 40 | 9.04 | 2.15 |
| 10:11 | 10 | 1982.28 | 165 | 2.31 | 105 | 45 | 9.04 | 2.22 |
| 11:12 | 5 | 4.5 | 41 | 0.12 | 21 | 7 | 9.94 | 4.77 |
| 11:12 | 6 | 18.12 | 66 | 0.34 | 33 | 11 | 7.5 | 3.89 |
| 11:12 | 7 | 71.47 | 81 | 0.37 | 47 | 16 | 9.49 | 3.02 |
| 11:12 | 8 | 319.65 | 134 | 0.67 | 62 | 31 | 9.49 | 3.09 |
| 11:12 | 9 | 652.37 | 164 | 1.33 | 87 | 47 | 9.49 | 2.72 |
| 11:12 | 10 | 2877.41 | 211 | 2.17 | 118 | 66 | 9.49 | 2.62 |
| 12:13 | 5 | 3.83 | 37 | 0.12 | 21 | 6 | 5.9 | 2.94 |
| 12:13 | 6 | 15.63 | 58 | 0.19 | 34 | 12 | 7.68 | 4.53 |
| 12:13 | 7 | 56.82 | 73 | 0.29 | 45 | 17 | 6.69 | 3.4 |
| 12:13 | 8 | 243.26 | 94 | 0.58 | 70 | 33 | 7.98 | 3.53 |
| 12:13 | 9 | 622.57 | 149 | 1.09 | 93 | 52 | 7.98 | 3.08 |
| 12:13 | 10 | 1534.14 | 160 | 1.68 | 112 | 62 | 7.98 | 3.02 |
| 13:14 | 5 | 4.67 | 39 | 0.16 | 19 | 1 | 6.52 | 6.52 |
| 13:14 | 6 | 18.3 | 61 | 0.19 | 34 | 7 | 6.52 | 3.01 |
| 13:14 | 7 | 57.6 | 72 | 0.35 | 48 | 11 | 6.52 | 2.05 |
| 13:14 | 8 | 175.38 | 104 | 0.64 | 73 | 25 | 6.52 | 2.57 |
| 13:14 | 9 | 507.44 | 125 | 1.51 | 101 | 43 | 6.52 | 2.24 |
| 13:14 | 10 | 1399.46 | 143 | 2.17 | 125 | 55 | 6.52 | 1.87 |

Table 6.4.5: Comparison between exact and no-mixing heuristic methods for 2-subsystem problems (Part II)

| | | Exact | | Heuristic | | | | |
|---|---|---|---|---|---|---|---|---|
| Subsystems | Cardinality | Time | Solutions | Time | Solutions | Dominated Solutions | Max REPOG (%) | Mean REPOG (%) |
| 1:3 | 5 | 27.22 | 34 | 0.28 | 11 | 1 | 3.17 | 3.17 |
| 1:3 | 6 | 174.78 | 64 | 0.28 | 19 | 5 | 5.86 | 3.15 |
| 1:3 | 7 | 833.05 | 104 | 0.66 | 34 | 12 | 5.86 | 2.4 |
| 1:3 | 8 | 9520.62 | 177 | 1.75 | 56 | 33 | 5.86 | 2.14 |
| 1:3 | 9 | 83803.67 | 247 | 2.2 | 83 | 49 | 5.86 | 1.8 |
| 2:4 | 5 | 36.37 | 43 | 0.14 | 14 | 3 | 2.75 | 2.01 |
| 2:4 | 6 | 126.95 | 65 | 0.27 | 18 | 5 | 3.31 | 2.29 |
| 2:4 | 7 | 786.3 | 110 | 0.7 | 40 | 18 | 3.54 | 2.04 |
| 2:4 | 8 | 11281.6 | 159 | 1.46 | 64 | 33 | 4.02 | 1.9 |
| 2:4 | 9 | 109604.22 | 221 | 2.48 | 94 | 49 | 3.67 | 1.83 |
| 3:5 | 5 | 36.73 | 44 | 0.17 | 17 | 4 | 10.63 | 5.5 |
| 3:5 | 6 | 125.74 | 67 | 0.41 | 21 | 10 | 8.16 | 3.76 |
| 3:5 | 7 | 664.04 | 93 | 0.86 | 45 | 26 | 10.47 | 2.86 |
| 3:5 | 8 | 12055.35 | 166 | 1.65 | 71 | 46 | 10.47 | 2.19 |
| 3:5 | 9 | 123965.16 | 236 | 2.52 | 92 | 60 | 10.47 | 1.97 |
| 4:6 | 5 | 21.16 | 42 | 0.12 | 14 | 4 | 8.5 | 2.78 |
| 4:6 | 6 | 93.08 | 40 | 0.44 | 27 | 5 | 7.78 | 3.61 |
| 4:6 | 7 | 760.86 | 75 | 0.95 | 54 | 17 | 7.78 | 2.42 |
| 4:6 | 8 | 7832.59 | 143 | 1.75 | 80 | 29 | 8.28 | 2.16 |
| 4:6 | 9 | 46553.37 | 98 | 2.66 | 112 | 33 | 7.31 | 2.08 |
| 5:7 | 5 | 15.19 | 46 | 0.12 | 14 | 3 | 1.16 | 0.39 |
| 5:7 | 6 | 139.92 | 47 | 0.29 | 27 | 6 | 6.74 | 3.11 |
| 5:7 | 7 | 607.83 | 80 | 0.73 | 49 | 15 | 6.74 | 2.01 |
| 5:7 | 8 | 8331.28 | 148 | 1.65 | 78 | 31 | 6.74 | 2.13 |
| 5:7 | 9 | 88083.28 | 154 | 2.2 | 110 | 40 | 6.74 | 1.93 |
| 6:8 | 5 | 23.04 | 42 | 0.16 | 15 | 2 | 6.29 | 3.57 |
| 6:8 | 6 | 192.83 | 91 | 0.29 | 35 | 15 | 6.53 | 3.25 |
| 6:8 | 7 | 1015.73 | 117 | 0.81 | 58 | 28 | 6.53 | 3.59 |
| 6:8 | 8 | 10861.82 | 169 | 1.45 | 83 | 50 | 6.53 | 2.58 |
| 6:8 | 9 | 85439.73 | 236 | 2.36 | 106 | 60 | 6.53 | 2.73 |
| 7:9 | 5 | 30.86 | 48 | 0.13 | 15 | 5 | 7.12 | 3.37 |
| 7:9 | 6 | 178.57 | 91 | 0.27 | 27 | 13 | 6.13 | 3.29 |
| 7:9 | 7 | 764.95 | 118 | 0.78 | 44 | 22 | 6.13 | 3.29 |
| 7:9 | 8 | 10045.38 | 178 | 1.39 | 69 | 36 | 6.13 | 2.55 |
| 7:9 | 9 | 95029.7 | 257 | 2.6 | 99 | 58 | 6.13 | 2.51 |
| 8:10 | 5 | 24.05 | 34 | 0.11 | 14 | 4 | 15.62 | 6.89 |
| 8:10 | 6 | 217.12 | 69 | 0.4 | 33 | 9 | 10.31 | 4.62 |
| 8:10 | 7 | 1331.66 | 146 | 1.05 | 57 | 25 | 8.92 | 3.83 |
| 8:10 | 8 | 6170.53 | 158 | 1.86 | 83 | 45 | 8.92 | 3.89 |
| 8:10 | 9 | 82987.19 | 255 | 2.86 | 122 | 73 | 9.77 | 3.19 |
| 9:11 | 5 | 34.24 | 43 | 0.13 | 14 | 1 | 2.09 | 2.09 |
| 9:11 | 6 | 121.59 | 73 | 0.31 | 23 | 11 | 7.74 | 3.12 |
| 9:11 | 7 | 648.55 | 99 | 0.87 | 41 | 20 | 7.19 | 3.31 |
| 9:11 | 8 | 8803.83 | 153 | 1.28 | 70 | 34 | 7.19 | 2.62 |
| 9:11 | 9 | 119520.06 | 244 | 2.37 | 100 | 46 | 6.26 | 2.43 |

Table 6.4.6: Comparison between exact and no-mixing heuristic methods for 3-subsystem problems (Part I)

| | | Exact | | Heuristic | | | | |
|---|---|---|---|---|---|---|---|---|
| Subsystems | Cardinality | Time | Solutions | Time | Solutions | Dominated Solutions | Max REPOG (%) | Mean REPOG (%) |
| 10:12 | 5 | 38.87 | 50 | 0.16 | 17 | 6 | 9.57 | 4.98 |
| 10:12 | 6 | 157.41 | 85 | 0.39 | 26 | 13 | 6.75 | 3.97 |
| 10:12 | 7 | 1394.77 | 134 | 0.8 | 44 | 26 | 6.75 | 3.33 |
| 10:12 | 8 | 7138.41 | 189 | 1.36 | 58 | 36 | 7.09 | 3.28 |
| 10:12 | 9 | 77682.65 | 282 | 2.03 | 80 | 50 | 6.75 | 2.94 |
| 11:13 | 5 | 29.67 | 38 | 0.12 | 20 | 1 | 0.0 | 0.0 |
| 11:13 | 6 | 177.08 | 74 | 0.36 | 37 | 16 | 8.62 | 4.31 |
| 11:13 | 7 | 803.57 | 113 | 0.98 | 58 | 40 | 9.3 | 3.78 |
| 11:13 | 8 | 5481.5 | 166 | 1.78 | 98 | 55 | 9.3 | 3.62 |
| 11:13 | 9 | 81353.39 | 193 | 3.39 | 137 | 82 | 9.36 | 3.25 |
| 12:14 | 5 | 25.08 | 46 | 0.11 | 19 | 4 | 6.63 | 4.17 |
| 12:14 | 6 | 278.38 | 84 | 0.26 | 36 | 16 | 6.63 | 3.29 |
| 12:14 | 7 | 814.25 | 128 | 0.89 | 58 | 36 | 6.41 | 3.41 |
| 12:14 | 8 | 11524.14 | 171 | 1.56 | 83 | 54 | 6.41 | 2.82 |
| 12:14 | 9 | 82984.34 | 253 | 2.57 | 116 | 73 | 6.41 | 2.66 |

Table 6.4.7: Comparison between exact and no-mixing heuristic methods for 3-subsystem problems (Part II)

### 6.4.4 No-Mixing Heuristic on Large Instances

We now wish to investigate how well the No-Mixing heuristic performs on large instances, by which we mean instances that cannot be solved exactly. To do this, we again construct problems by forming systems out of contiguous subsystems from the original problem, keeping all the different component types, and again just considering the cardinality knapsack constraint, which will be set equal to 3 times the number of subsystems. For example, the problem instance 2:5 considers a system composed of subsystems 2, 3, 4 and 5 from the original problem set, and the cardinality constraint will have a right-hand-side value of 12. We will consider all such problem instances $a : b$ with $1 \leq a \leq b \leq 14$. Throughout, within the No-Mixing heuristic we again update the $\varepsilon$ constraint after each solve by setting $\varepsilon = 0.99 \cdot LHS$, as described in the previous section. To visualise these results, we plot the number of subsystems in a given problem instance against the runtime in Figure 6.4.1, and against the number of solutions in the Pareto front in Figure 6.4.2.

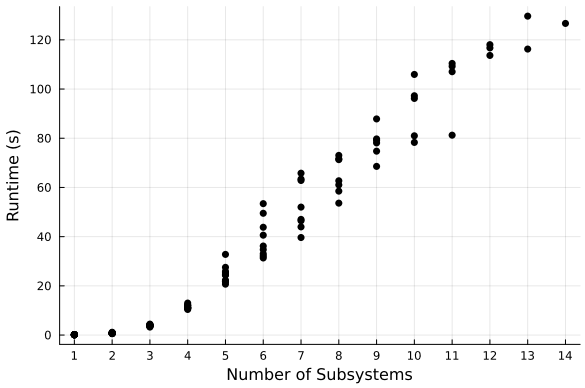



Figure 6.4.1: Size and runtimes of problem instances

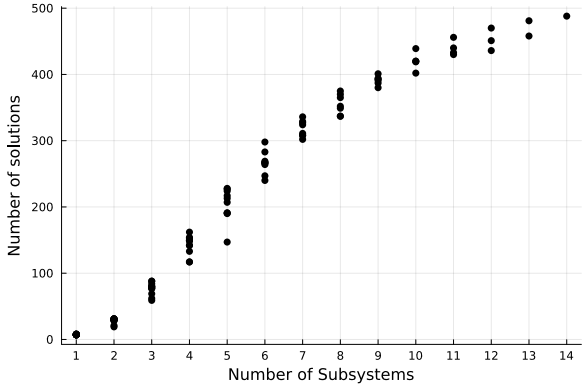



Figure 6.4.2: Size and number of solutions of problem instances

Figure 6.4.1 shows a roughly linear increase in runtime as the number of subsystems increases. This is roughly to be expected, as the column generation phase of the heuristic takes far more time than the RMP stage, and this column generation phase will naturally increase linearly in runtime as the number of subsystems increases. Figure 6.4.2 then shows a sub-linear increase in the number of Pareto-optimal solutions as the number of subsystems increases. This is likely due to the use of the $\varepsilon$-constraint method with the updating rule that we specified. Trying it again with the more careful update rule $\varepsilon = 0.999 \cdot LHS$ on one of the instances did show that solutions were being missed; however, this of course took longer to run due to performing the extra solves of the RMP for the extra values of $\varepsilon$. We can also argue that these extra solutions would not provide any more real-world flexibility to a decision maker, as the original update step already yields Pareto fronts with hundreds of solutions.

## 6.5 Conclusions

In this chapter, we combined aspects from both Chapter 4 and Chapter 5 to investigate the integrated design and dynamic maintenance of series-parallel systems. To construct the optimisation model, we used both the decomposable impulsive CTMDP framework of Chapter 5 alongside our MDP Design framework as outlined in Chapter 3 and applied

in Chapter 4. While we know that this model is intractable in terms of obtaining exact solutions, we still need a way to get exact solutions for small problem instances for the sake of comparison against heuristics. To this end, we developed a bottom-up dynamic programming approach for constructing the state, action, and state-action spaces of the maximal CTMDP for the MDP Design problem. We have also proposed a detailed decomposition-based solution methodology for this problem based on column generation and linearisation by converting nonlinear problems to bi-objective linear problems. We then tested the extent to which large problems can be solved exactly, and found that even with only one objective function, the model grew to be unsolvable by a MILP solver very quickly. We then tested a simple version of our heuristic, the No-Mixing heuristic, against the exact solver for problems that only specify one component type per subsystem. This yielded encouraging results in terms of the size and quality of the heuristic Pareto fronts. The No-Mixing heuristic also scaled well to larger problem instances, where exact solutions could not be found in reasonable time nor within reasonable amounts of memory.

While this chapter has offered a promising solution methodology and encouraging initial results, it remains to be seen just how well the full methodology performs in terms of both scalability and solution quality. There are multiple times throughout the methodology where we suggest that there is uncertainty in how many of the generated columns should be retained, and whether or not the solution methods for pricing subproblems should terminate early if a negative reduced cost column is found. This is especially important considering the fact that the column generation methods solve bi-objective problems to produce single solutions, meaning many additional columns could be retained, and this would lead to a more flexible yet more difficult RMP. However, these bi-objective methods could be more computationally intensive, and early termination could lead to a far faster heuristic. Many variations of the column generation heuristics would need to be thoroughly investigated to find their strengths and

weaknesses. Depending on the performance of these heuristics, it may also be worth investigating the various extensions discussed, such as cutting planes and branch-and-price. Another difficulty that will have to be dealt with is the fact that problem instances that do allow for component mixing become large very quickly, leaving us with no exact point of comparison for our heuristics. One option could be to simply investigate the extent to which the column generation methods improve upon the no-mixing heuristic, and the cost of this in terms of running times. Finally, it is worth noting again that at the core of our proposed column generation methods is a heuristic, the APP method, for an MDP Design pricing subproblem. The creation of an efficient exact solver for such problems remains an important direction for future research.

# Chapter 7

# Conclusions and Further Research

In this thesis, we have explored the intersection of combinatorial, dynamic, and bi-objective optimisation in the context of systems with redundancy. In this chapter, we summarise our achievements, drawbacks, and findings in Section 7.1, and explore potential avenues for further research in Section 7.2.

## 7.1 Conclusions

In Chapter 4, we introduced an extension to the single-subsystem RAP that incorporates a long-run operation cost objective made up of usage and maintenance costs, alongside the usual reliability objective. We also integrated dynamic maintenance and system design into a unified MILP optimisation model, following our MDP Design framework from Chapter 3. The resulting model contains an intractable number of variables and constraints due to the need to represent all possible states across all possible designs, and requires being solved many times due to its bi-objective nature, necessitating the creation of a bespoke solution methodology for this type of problem. To this end, we developed the heuristic APP methodology, which allows us to decompose a bi-objective MDP Design problem into a design phase and an MDP phase. The application of APP to our design-and-maintenance problem yielded a bi-objective

supermodular optimisation problem, which we then scalarised using the $\varepsilon$-constraint method, and linearised using the probability chains method of O'Hanley et al. (2013). Via numerical experiments, we found that APP performed very favourably against an exact MILP solver, both in terms of the quantity and quality of solutions on the Pareto front, and in terms of the massively reduced computational time.

In terms of managerial insights for the problem, we noted that the majority of solutions only used two types of component, keeping the complexity of optimal system designs (and their resulting MDPs) quite low. This could be roughly interpreted as having a primary component type with a more favourable usage cost, and a secondary component type with a higher but less important usage cost (as this cost would only ever be incurred when all copies of the primary type were damaged). Additionally, we noted that the Pareto fronts for the integrated problem were far better populated than the corresponding Pareto fronts for the design-only problem, demonstrating that the incorporation of dynamic maintenance allows for far greater flexibility for decision makers in terms of determining a trade-off between costs and reliability. On top of this, we also identified some cases where design solutions that were Pareto optimal for the design-only problem were in fact Pareto dominated by a solution that used a dynamic policy. This was more common in systems with more redundancy. Of course, this work was not without its weaknesses, which offers room for further research. The modelling assumptions were quite simplistic, such as only considering a parallel system as opposed to series-parallel or complex, and the components in the system worked with active redundancy (i.e., no difference in failure rate depending on whether or not the component is being used), and failure and repair times were both assumed to be exponentially distributed. Additionally, while the APP methodology was proved to have a bounded optimality gap in general, a specific gap was not derived for the specific problem at hand. Due to the high-dimensional nature of the maintenance MDP, we expect that doing so would be extremely difficult.

In Chapter 5, we developed the mathematical machinery necessary to extend the MDP model of Chapter 4 to the series-parallel case, and developed a solution methodology for this type of model. We began by generalising, formalising, and analysing the implementation of impulsive control (e.g. instant transitions) that was applied in Chapter 4. Notably, we used equivalence relations to analyse this framework, and proved that a particular class of intuitive policies (the coherent policies) always contain an optimal solution. We also demonstrated that it is always possible to obtain a coherent optimal policy from a non-coherent one. We then extended this impulsive CTMDP model to the decomposable impulsive CTMDP, a model which can be thought of as multiple separable impulsive CTMDPs which can be controlled together, and which are unified by shared objective functions. We again conducted the necessary analysis to prove that, when controlled by a separable coherent policy, the resulting Markov process decomposes into independent Markov processes for each compartment, up to an equivalence relation. This fact allows us to heuristically block-diagonalise the LP formulation of a decomposable impulsive CTMDP, allowing for the use of Dantzig-Wolfe decomposition to solve the problem. This in turn allows us to develop a heuristic in which we find Pareto fronts for the compartment CTMDPs that made up the decomposable impulsive CTMDP, and then piece these solutions back together with a bi-objective binary program, scalarised using the $\varepsilon$-constraint method.

In the numerical study, we applied our framework and solution methodology to the problem of dynamically maintaining a series-parallel system. The problem was formulated as a decomposable impulsive CTMDP made up of compartment CTMDPs for each subsystem. Using toy examples of this model, we compared the performance of an exact MILP solver against our decomposition-based heuristic. We found that the heuristic performed well in terms of the quality and quantity of the solutions, however not quite as favourably as the heuristic we used in Chapter 4. This heuristic encountered more problem instances where many solutions on the heuristic Pareto front were

slightly dominated by solutions on the exact Pareto front, or where individual heuristic solutions were more heavily dominated by an exact solution. We suspect that this is due to the relative inflexibility of the separable policies to which the heuristic is limited. This inflexibility is likely also the reason that the heuristic returned far fewer solutions than the exact method. However, as problem sizes scaled up, the number of solutions from both the exact and heuristic methods increased, and Pareto fronts from the heuristic method on even moderately sized instances can qualitatively be said to offer a great deal of flexibility to the decision maker, despite technically being a fraction of the solutions that would be returned by the exact method. While the heuristic did face these minor shortcomings, across the board we found that it performed very well in terms of computational time. Notably, it solved 4-subsystem instances, which took on the order of three hours by the exact method, in less than half a second. On a 14-subsystem problem, the scale of which could never be solved by the exact method in reasonable time, the heuristic took only five seconds and returned a Pareto front with over 250 solutions. A shortcoming of this work is that our implementation of impulsive control is not as feature-rich as previous implementations, notably lacking multiple gradual actions, instant lump costs, and random impulsive jumps. However, our framework was enough for the problem that we were interested in. Filling these gaps offers an interesting direction for further research. We also did not provide any sort of theoretical optimality bounds on separable policies against general policies for decomposable impulsive CTMDPs. We expect that this would be quite difficult for single non-linear objective functions, and even more difficult for multi-objective problems.

Chapter 6 merged the difficulties of integrated design and maintenance from Chapter 4 with the high-dimensional yet decomposable nature of the series-parallel systems from Chapter 5. We provided a MILP formulation; an efficient method for constructing the spaces of states, actions, and state-action pairs; and outlined a detailed heuristic. We then demonstrated how quickly the model becomes impossible to solve exactly, even

with our efficient model construction, and compared the exact solver against a partial implementation of our heuristic for small, solvable problem instances, yielding encouraging results. We also demonstrated that the heuristic scales well to larger problem instances which would be impossible to solve exactly. Of course, this partial implementation does not tell the whole story, and more work needs to be done in implementing the full methodology to see how well it performs. Work also needs to be done on how aspects such as usage and repairs costs affect the solutions of a particular problem, as we investigated in Chapter 4. As with Chapter 4 and Chapter 5, our use of heuristics still leaves a lot of room for the development of exact solution methodologies, or improved heuristics, for problems of this type.

Pulling all of this together, we have developed a unified modelling framework for the integration of strategic and operational decision-making with multiple objectives, and applied this to the problem of joint design and dynamic maintenance optimisation for redundant systems under the competing objectives of cost and reliability. We began with a design-and-maintenance problem for a parallel system, extended the maintenance-only problem to the series-parallel case, then merged the two together to obtain an integrated design-and-maintenance problem for series-parallel systems. This has required a combination of modelling and solution techniques from the areas of Markov decision processes and mixed-integer optimisation, as well as the construction of bespoke decomposition-based heuristics to address problems that are otherwise computationally intractable. This use of decomposition has allowed us to exploit problem structure so as to drastically reduce computational burden while preserving high-quality approximations to the true Pareto front. However, the identified modelling simplifications, restricted policy classes, and absence of tight optimality guarantees indicate important avenues for future research. The following section concludes with an outline of many possible directions for further research.

## 7.2 Further Research

### 7.2.1 MDP Design

In this thesis, we have introduced and applied our own MDP Design framework, allowing for the formulation of problems in which a system is first designed in a strategic phase, and then controlled during an operational phase. Specifically, we applied this framework to the problem of designing and maintaining redundant systems. It is clear that there is much work to be done with MDP Design problems, in terms of theoretical development, optimisation, and application to real-world problems. While we have formulated a flexible MDP Design framework in such a way that it can be modelled and solved as a MILP, we know that such exact methods will not scale well due to the fact that all state-action pairs across all designs must be included in the MILP. To this end, we introduced the APP framework, which serves as a heuristic. However, work still needs to be done on the development of an efficient exact solver for MDP Design problems. A likely candidate for this is Benders decomposition, due to its well-known applicability to two-stage problems. For a continuous LP of the form $\min_{x,y \in \mathcal{X}\mathcal{Y}} \left\{c^T x + d^T y\right\}$, we can express this problem in just the $x$ variables as $\min_{x \in \mathcal{X}} \left\{c^T x + d(x)\right\}$, where $d(x) = \min_{y \in \mathcal{Y}(x)} \left\{d^T x\right\}$. Then, the function $d(x)$ can be replaced by a variable $z$ and then represented by a finite set of inequalities which capture the value of the function (called optimality cuts) and which capture the domain of the function (feasiblity cuts), as some first-stage variables may result in infeasible second-stage problems. Benders decomposition solved this type of two-stage problem by repeatedly generating optimality and feasibility cuts by using dual information from the second-stage LP. Extending this to binary first-stage variables, Laporte and Louveaux (1993) introduced an algorithm in which the problem is solved via branch-and-bound, and each node of the tree is solved via Benders decomposition. The generalisation of this algorithm has since become commonly known as "branch-and-Benders-cut". At first, this seems applicable

to our MDP Design framework due to our first-stage binary variables and second-stage continuous variables; however, applicability is not immediate, due to Benders' reliance on dual variables for the linear program. Indeed, the main source of difficulty for our problem is the exponentially large number of continuous decision variables, so solutions that require information from the whole linear program are not suitable. However, in the same work, Laporte and Louveaux (1993) introduce a separate class of Benders cuts for integer-feasible first-stage solutions which do not require dual information, and instead only require an initial lower bound on the second-stage solution, and for the second-stage objective function to be computable for any feasible first-stage solution. This could be applied to the MDP Design problem by repeatedly solving the Benders decomposition master problem to optimality, and adding new cuts of this type at the incumbent optimal solution. The hope would be that this approach converges after probing only a reasonable subset of the possible design solutions. This could also be adapted to be used as a heuristic for the case where exploring the design space (i.e., solving the Benders master problem) is relatively easy, but solving the MDP is hard; in this case, the MDP for a given design can be solved approximately, and corresponding approximate cuts can be added based on those solutions. The bi-objective version of this problem could also use this method by building it into the dichotomic method. For any weighted objective node being tested in the dichotomic method, this node can be solved via our proposed solve-and-cut method. The performance of this could be improved by finding the *Pareto front* of any candidate design (rather than just the mixed-objective solution) and memo-ising it for later solves, meaning we wouldn't have to solve it from scratch each time. Additionally, any new node can be "warmed-up" by initialising it with cuts based on these previous solutions, rather than solving it from scratch. An efficient way of solving problems with three or more objectives remains an open question (as it does in general for multi-objective optimisation).

While the development of an exact solver for MDP Design problems is an important

and necessary stepping stone in improving the understanding of this class of problem, such a solver will be limited in its applicability: it is well-known that MDPs suffer from the curse of dimensionality, let alone MDP Design problems. This fact necessitates the further development of heuristic solvers, both problem-specific and general. Our APP framework could continue to serve as a useful baseline for problems where a reasonable design problem (SP1) can be formulated and solved, however its strength in problems outside of those that we've presented remains to be seen. In terms of general heuristic solvers, an interesting direction of research would be the integration of reinforcement learning within a metaheuristic. This would allow the metaheuristic to explore the combinatorial design space, with the reinforcement learning algorithm being used to optimise the MDP for any given design. While such an approach should in theory be able to tackle complex real-world problems, an immediate problem is that it would require jointly tuning the hyperparameters of both the metaheuristic and the reinforcement learning algorithm. If *deep* reinforcement learning was used, one would either: have to determine hyperparameters and neural network architectures that work well for any potential design, or; have to somehow tune hyperparameters and neural network architectures within the metaheuristic search. In either case, both exact and heuristic optimisation for MDP Design problems remains an interesting direction for future research.

While the MDP Design framework we have presented is very flexible in terms of the relationship between design decisions and state-action feasibility, it does not incorporate some of the features of the framework of Brown et al. (2024). Namely, our framework could be extended so that the state-action cost functions can be directly affected by design decisions without the need to introduce entirely new state-action pairs, and could additionally be extended to include uncertainty in the parameters of the MDP that results from any given design. The latter is of particular interest for reliability, as the components ordered from a manufacturer may vary from the manufacturer's stated

reliability. Such an extension could be achieved by "merging" the MILP formulation of our framework with the single-level MILP reformulation of the bi-level model found in Brown et al. (2024). However, the authors of that paper note that solving the single-level model as a MILP does not perform as well as solving the bi-level model using a bi-level solver. As such, solving the resulting model would itself be an avenue for future research.

An issue briefly noted in Chapter 4 is the failure of both mixed-objective and $\varepsilon$-constraint scalarisations of the MDP Design MILP to easily return all Pareto-optimal solutions corresponding to deterministic policies. The mixed-objectives method, as with any MILP, can only return supported solutions, and therefore returns an incomplete Pareto-front. On the other hand, the $\varepsilon$-constraint method may return undesirable randomised policies. In order to return the Pareto front of design-policy solutions with deterministic policies, where each solution is non-dominated with respect to all design-policy solutions with randomised policies, a different solution methodology will need to be developed. For a $K$-objective problem, the $K-1$ $\varepsilon$-constraints will result in at most $K-1$ randomisations, and thus at most $2^{K-1}$ ways to "round-off" or collapse the randomised policy into a deterministic policy. For small $K$, it seems reasonable to suggest that all such deterministic policies could be checked, and that at least one of these solutions would be Pareto-optimal. Alternatively, the $\varepsilon$-constraint method could be used to find all designs that correspond to some Pareto-optimal solution, and then find the Pareto front of all MDPs corresponding to these designs, along similar lines to SP2 in our APP methodology. Indeed, in Chapter 4, we demonstrated that the APP method was able to return solutions not found by the exact method under a mixed-objective scalarisation; however, we did not verify that they were truly Pareto-optimal, only that they were not dominated by any supported solutions from the exact method. Practically speaking, wherever supported solutions from the mixed-objectives method provide a flexible enough set of solutions for a decision maker, or wherever "rounded"

randomised polices from the $\varepsilon$-constraint method appear to be "good enough", this problem isn't particularly worrying. However, more research is needed to fully understand how to properly obtain unsupported Pareto-optimal solutions.

### 7.2.2 Applications

Future research should consider different potential applications for the MDP Design framework. Even within the topic of joint design and maintenance, the work done in this thesis could be extended from simple systems of components to networks of critical infrastructure, such as gas or water pipelines. For example, Alderson et al. (2015) considers a defender-attacker-defender model for the reinforcement (design) and maintenance of a network of pipelines against an intelligent attacker. Such a model could be extended to consider the dynamic maintenance of such a system, although the resulting problem would be very difficult. Of course, the applicability of the MDP Design framework goes far beyond design and maintenance. As noted in Section 4.5, one could consider a queueing control problem in which the design variables select from a range of servers, and the MDP models the problem of routing incoming customers to the servers chosen in the first stage. Alternatively, the design variables could correspond to the assignment of limited floor space to different queues which serve different types of customers, resulting in an MDP which models the control of a capacitated queueing system. An example of this could be the check-in area of an airport, where limited floor space is available to many different airlines, who in turn use a mixture of manned check-in desks and self-serve kiosks. Design variables could include the total number of manned check-in desks installed and their assignment to different airlines, as well as the assignment of remaining floor space to different airlines for the use of self-serve kiosks. The MDP could then model the control of the queueing process, for example encouraging customers to use either manned desks or kiosks. Alternatively, there may be no dynamic control available, and any given design would simply correspond to a

Markov chain to model the queueing process, which the MDP Design framework is also suited for (a design may correspond to only one action per state). On the theme of airport design and control, similar problems could be modelled for security and passport control. For the former, one may consider design elements such as the number of paid "fast-track" queues, and control elements such as sending individuals or groups (the latter important for families) to security queues. For the latter, in the context of an EU airport for example, design elements may include the installation of EU and non-EU passport lines, as well as determining the mixture of manned and automated passport checking. Control could potentially allow non-EU citizens into EU queues or vice versa in the case of an imbalance, or open and closed manned passport desks.

Another important control problem with aspects of queueing is that of hospitals. Hospitals can often be modelled as queueing systems, and can also have an aspect of admission control over patients, especially when the patient is not facing an emergency. New hospitals must be designed to fit the needs of the surrounding area, and will always result in this type of admission control problem, and as such this problem also fits within the MDP Design framework. It is reasonable to suggest that the queueing aspect won't have much of an impact on the optimal design, as data on the local population and the availability of services in other nearby hospitals will be the primary determinants of the allocation of resources; however, it may still be worthwhile to consider these problems together.

A problem commonly modelled as an MDP which could be considered to have design elements is that of inventory control, where decisions on when to place an order for an item, and how large that order should be, are taken based on the remaining stock-on-hand and outstanding previous orders. At a store level, design decisions have to be taken to design the "back of the store" where items will be stored before being taken out to the shelves open to the public; this would include decisions on the number of fridges, freezers, and space for large and small items. One could go even further and

consider the joint design of the whole store (i.e., both private storage and public storage) alongside inventory optimisation. At a higher level considering multiple stores ordering from warehouses, design elements could include the opening of new warehouses, their locations, and the assignment of stores to warehouses; the inventory policy of all stores could then be optimised based on those design choices.

Related to opening warehouses is the opening of depots which store both goods, service providers, and vehicles. This situation gives rise to the vehicle routing problem (VRP), where vehicles need to be routed to serve customers in different locations and then return to the depot. Two existing extensions of this are: the location-routing problem, where depots are placed and routes are planned in an integrated problem; and the dynamic vehicle routing problem, where routes have to be updated dynamically in response to new customer requests or alterations in traffic conditions. These problems are completely separate in the literature; however, the MDP Design framework could be used to integrate them into a single optimisation problem, where the placement of depots and the dynamic routing (and re-routing) of vehicles is simultaneously optimised. This is very much an example of a problem that would require the development of heuristics, as the dynamic VRP alone cannot be solved exactly and instead relies on approximate methods.

A final suggestion for an application area is that of platform trials, which is a type of clinical trial in which different "arms" (treatments) can enter and leave the trial over time. Platform trials are operated dynamically to manage patient outcomes and improve the statistical power of the data collected, and in this sense may be seen as a multi-objective problem, where decisions include the allocation of patients to arms, and the inclusion of new arms and exclusion of poorly performing arms. However, platform trials are also subject to an initial design phase, namely the design of the "master protocol": the set of rules by which the trial will abide in order to follow the law, ensure safety, and ensure that data is collected in a consistent manner so that the final

set of data can be properly statistically analysed. Additionally, the design phase must determine the hospitals (as well as pharmacies and other facilities) that will participate in the trial, and the extent to which they will participate. The goal would be to design and operate a trial that is cost effective, good for patients, and which obtains a strong set of data for the final analysis.

All of the suggested application areas will no doubt yield their own interesting problems, and push our proposed framework to its limit or even beyond its breaking point. This is the type of stress test necessary to further develop both the framework and its solution methodologies, and to ultimately improve real-world decision making.

### 7.2.3 Impulsivity

In Chapter 4, we used the notion of impulsivity to allow state-action pairs to act as though they had instantaneously transitioned into another state, under that state's zero action. We did this to model the fact that the time taken to decide to do a repair and dedicating the process to that repair (i.e., calling the necessary engineers) is negligible (and can therefore be modelled as instant) in comparison to the time taken to do the repair, and the time taken for components to fail. In Chapter 5, we expanded upon and generalised our implementation of impulsivity to ensure that it behaved as we expected. However, this implementation misses many of the features offered by the implementation of impulsive control given by Dufour and Piunovskiy (2015). An interesting research direction would be to extend our model to include some of those features, whilst keeping the relative mathematical simplicity and compatibility with pre-existing solution methods of our model.

The most reasonable extension would be the incorporation of multiple gradual actions for each state. Wherever one could take an impulsive action $a^I$ followed by gradual action $a^G$ in state $s \oplus a^I$, the CTMDP would instead have a joint action $(a^I, a^G)$ to model this, again representing a sequence of actions through a single joint action. Equivalence

classes over states and state-action pairs would be changed to account for the two types of action: roughly, state-action pair $(s_1, (a_1^I, a_1^G))$ would now be impulsively equivalent to $(s_2, (a_2^I, a_2^G))$ if $s_1 \oplus a_1^I = s_2 \oplus a_2^I$ and $a_1^G = a_2^G$. We expect that changes of this sort would "ripple through" the theory of Chapter 5 quite smoothly, and allow for a more versatile model.

A more difficult extension would be the inclusion of lump-sum costs. For any $(s_1, a_1), (s_2, a_2)$ such that $s_1 \oplus a_1 = s_2 \oplus a_2$, it may be the case that the lump-sum costs of the former and the latter disagree. In other words, two state-action pairs may instantly jump into the same state, but at different costs. In this case, the "cost-equivalence" of these two pairs breaks down, as does much of the equivalence theory as developed in Chapter 5. As such, more work would have to be done to ensure that notions such as coherent optimal policies continue to remain true.

The most difficult extension would be that of stochastic impulsive transitions, that is to say impulsive actions which may instantly jump into a range of states probabilistically. In this case, our notion of coherence breaks down, as there is no longer a direct correspondence between actions and target states (i.e., we cannot simply target "good" states, and may be forced to take a sequence of jumps if we impulsively transition into undesirable states). It is unclear whether or not this idea can easily be incorporated into the standard CTMDP in the way we have done for our more simple model of impulsivity. However, the applicability of this modelling feature is also unclear, and is not discussed by Dufour and Piunovskiy (2015). To the best of our knowledge, two (toy) examples of impulsive CTMDPs are known to exist: the immunisation problem of Piunovskiy (2004), and the fluid control problem of Piunovskiy and Zhang (2025), and both of these only allow for deterministic jumps. It remains to be seen whether any realistic problem truly requires the random jumps which the framework of Dufour and Piunovskiy (2015) allows.

### 7.2.4 Redundancy Allocation

Throughout our work, we have focused on redundant systems and redundancy allocation. While we introduced some novel notions such as usage costs and component-wise repair costs, other aspects of our models remained simple: only one component per subsystem was ever required, only hot standby (or active redundancy) was considered, system designs were simple parallel or series-parallel designs, the number of ongoing repairs at any given time was unconstrained, components followed simple exponential distributions for failure and repair times, and component switching was assumed to be perfect. Each of these simplifications offers opportunities for further research, which we shall address in turn.

The assumption of only requiring one component per subsystem can be generalised to so-called $k$-out-of-$n$ subsystems, in which $k$ components are required to be healthy. One could imagine the system to fail entirely if less than $k$ components are available, or the system may simply lose some capacity. From the maintenance point of view, this would be a simple extension of the CTMDP models we have presented in this work. From the design-and-maintenance point of view, a reasonable approach would be to use APP and try to extend the probability chains method employed in Chapter 4 to the $k$-out-of-$n$ case to obtain system designs. One would expect the range of Pareto-optimal policies to be less flexible, due to the greater need to keep more components active simultaneously. This can be further generalised to weighted-$k$-out-of-$n$ systems, where we imagine some flow moving through the system, and each component has some flow capacity, and all flow can only be processed if the total capacity of the healthy components exceeds the flow. The associated maintenance problem would again be a simple extension of our CTMDP models; however, the design problem may prove more difficult due to the many combinations of component capacities that do and don't meet the flow requirements. All of this can also be extended to *consecutive* $k$-out-of-$n$ systems, which are the $k$-out-of-$n$ generalisation of series-parallel systems.

As well as the hot-standby redundancy using in this work, components can also be used in a warm- and cold-standby setting, where the former refers to a situation where idle components face a lower rate of failure than if they were being used, and the latter refers to the situation where idle components face no probability of failure. Both of these can be considered more realistic than hot-standby, as it may be unrealistic to assume that an idling component faces the same failure rate as if it were being used. One can also employ mixed strategies, where components are kept in a mixture of hot, warm, and cold standby. The decisions to have components in hot, warm, or cold standby would have to be reflected in the design phase of the problem, and would of course affect the stochastic dynamics of the MDP stage. An early version of our MDP model did in fact model warm-standby by having a different exponential rate of failure for one copy of the cheapest available component; however, analysis of this model proved difficult in terms of finding a nice closed form for the long-run average cost under a simple maintenance policy. Unless such a closed form could be determined, this difficulty would prevent the use of the APP methodology, as a simple design phase would not be possible. This, in turn, would further motivate the development of different (and perhaps exact) solution methodologies.

Extending to complex systems, i.e., general networks of components, can be expected to introduce significant non-linearities in the reliability function to capture the dependency structures of the system, as well as drastically increasing the dimensionality of the MDP. A reasonable approach would be to continue to use decomposition based methods to focus on designing and maintaining the individual subsystems, and then piecing them back together. However, unlike the reassembly of subproblems in Chapter 5 and Chapter 6, the reassembly in the general network case is non-trivial. An additional complication is that general networks can have *implicit* redundancy as well as the explicit redundancy explored in this thesis; for example, two or more entirely independent routes in a network between source and terminal nodes. As such, decom-

posing into subsystems with explicit redundancy could miss such subtleties. Solving this problem may require more graph-theoretic approaches.

Limitations on repair resources (i.e., on the number of simultaneous ongoing repairs) is yet another extension that would affect the MDP, this time by constraining the action spaces. This in turn complicates the formulation of closed-form long-run average costs, again potentially limiting the usefulness of APP. For a single subsystem, the maintenance problem itself is no more complex than the MDP presented in Chapter 4, and could be solved exactly. In the series-parallel case, the notion of a decomposable MDP with global linking constraints on the action space seems closely related to *weakly-coupled* MDPs (Meuleau et al., 1998), although such MDPs are usually linked by a single linear objective function rather than multiple objectives with non-linearities, and they are normally done in discrete time. As such, research on the maintenance of a series-parallel system under repair constraints would necessitate the further development of weakly-coupled MDPs in general.

We now turn our attention to the modelling of the degradation of components. Our models have used simple ternary-state models where components are either healthy, repairing, or damaged, under the assumption of exponentially distributed sojourn times of transitions between these states. A simple extension of this would be to use multi-state models, where components go through multiple finer grain levels of degradation before failing, with transitions remaining exponentially distributed. An example of previous work using such a modelling approach would be Tavana et al. (2018), who considers an RAP with repairable multi-state components. Components modelled in this way can still be assigned a long-run reliability value, so the design objective of APP remains applicable. However, the MDP becomes more complicated due to a far increased dimensionality. Decomposition could again be applicable here, as the system-wide MDP could be decomposed into many one-component MDPs. However, this would lose the system- and subsystem-wide strategies employed in our work so far, where components

are sometimes left damaged until other components have also become damaged. It could potentially be possible to do the following: (1) find maintenance Pareto-fronts (maintenance costs and reliability as opposed objectives) for each component types in a problem; (2) treat each component-maintenance-strategy pair as different component types with different reliabilities and (average) maintenance costs (but equal usage costs and constraint contributions) in a design problem; and (3) approximate these components as having exponential degradation times and some sort of constant average maintenance cost rate in a system-wide MDP to determine the system-wide maintenance strategy. The first and second parts of this approach can be thought of as a dynamic-maintenance-based and discrete-valued version of the reliability-value selection found in the reliability-redundancy allocation problem (RRAP), which is a variation of the RAP in which component reliability is itself a decision variable (Kim and Kim, 2017). This approach would approximately decompose the problem into smaller subproblems which can more readily be solved. Of course, multi-state models are also limited due to their assumption of memoryless transitions between states. The component lifetimes or degradation are in fact better modelled as being Weibull-distributed or modelled via a Gamma process Van Noortwijk (2009). However, such models are not well-suited to fit within the MDP framework, and a great deal of mathematical care would need to be taken when dealing with non-memorylessness within a decision process.

The final area for extensions of our RAP model would be the introduction of imperfect switching, where swapping from an in-use component to a different one upon its failure may not succeed. We noted in Chapter 4 that this could be included in the MDP model by adding extra decisions on which component to switch to upon failure. This would again complicate or prevent simple long-run average formulae for the design phase, and would add a small amount of complexity to the MDP. Due to the former, such a problem would again necessitate the need for the further development of solution methodologies for MDP Design problems.

# Appendix A

# Appendix for Chapter 4

## A.1 Section 4.2 Supplement

### A.1.1 Weakly Communicating and Not Unichain

Here, we provide the proof regarding the weakly communicating but not unichain properties of DMP.

**Theorem A.1.1.** *DMP is weakly communicating for $N > 0$. However, there exist $N$ and $M_i$ for $i = 1, ..., N$ such that DMP is not unichain.*

*Proof.* We first show that DMP is weakly communicating. Let $\mathcal{S} = \bigtimes_{i=1}^{N} \{(s_1, s_2) : s_1 + s_2 \leq M_i\}$, and choose two states $\boldsymbol{s}, \boldsymbol{s}' \in \mathcal{S}$. Assume for now that state $\boldsymbol{s}'$ contains at least one healthy component. We choose an action $\boldsymbol{a} \in \mathcal{A}(s')$ as follows:

$$[\boldsymbol{a}]_i = M_i - [\boldsymbol{s}']_{i,2}.$$

We then define the policy $\mu$ as follows:

$$\mu(\boldsymbol{s}) = \begin{cases} \boldsymbol{a}, & \boldsymbol{s} = \boldsymbol{s}_F, \\ 0, & \text{otherwise.} \end{cases},$$

where $\boldsymbol{s}_F = ((0, M_1), ..., (0, M_N))$ is the completely failed state. Let $p_\mu^m(\boldsymbol{s}, \boldsymbol{s}')$ be the probability of reaching state $\boldsymbol{s}'$ from state $\boldsymbol{s}$ after $m$ transitions under policy $\mu$. Clearly, $p_\mu^m(\boldsymbol{s}, \boldsymbol{s}_F) > 0$, for some $m > 0$ due to the inactivity of policy $\mu$, and $p_\mu^n(\boldsymbol{s}_F, \boldsymbol{s}') > 0$ for some $n > 0$, as with non-zero probability after taking action $\boldsymbol{a}$ in $\boldsymbol{s}_F$, the suitable subset of components which are healthy in $\boldsymbol{s}'$ all finish repairing before the components which are still repairing in $\boldsymbol{s}'$. Therefore $p_\mu^{m+n}(\boldsymbol{s}, \boldsymbol{s}') > 0$.

Assume now that state $\boldsymbol{s}'$ has no healthy components, so all components are either damaged or currently being repaired. Taking action $\boldsymbol{a}$ in state $\boldsymbol{s}_F$ would "leapfrog" state $\boldsymbol{s}'$ due to impulsive actions, so we amend action $\boldsymbol{a}$ by starting repairs on an additional component (assuming that this exists). Then, with non-zero probability, this additional component both finishes repairing first and immediately fails again, leaving the desired components still repairing.

All that is left now is the consideration of state $\boldsymbol{s}_R = ((M_1, 0), ..., (M_N, 0))$, the all-repairing state. In this state, there is no "additional" component to repair, as all components are currently being repaired. In fact, this state is transient under all policies, as transitions from any state under any action represent either: a repair completion, so the next state must have at least one healthy component; or a new degradation, meaning the next state must have at least one damaged component. Therefore, $\boldsymbol{s}_R$ is transient under all policies, meaning DMP is weakly communicating as $p_\mu^m(\boldsymbol{s}, \boldsymbol{s}') > 0$ for all pairs of states $\boldsymbol{s}, \boldsymbol{s}'$ for some policy $\mu$, except for $\boldsymbol{s}' = \boldsymbol{s}_R$, which is uniquely transient under all policies.

It remains to show that DMP is not unichain for some $N$ and $M_i$. Consider a DMP

with $N = 2, M_1 = 1, M_2 = 2.$, and a policy satisfying the following:

$$\mu((0,1),(0,2)) = (1,0)$$
$$\mu((0,0),(0,2)) = (0,0)$$

$$\mu((0,1),(1,0)) = (0,0)$$
$$\mu((0,1),(1,1)) = \mu((0,1),(0,1)) = (0,1)$$
$$\mu((0,1),(0,0)) = \mu((0,1),(2,0))) = (0,0).$$

Suppose we start in state $((0,1),(0,2))$, then it is clear to see we have recurrent class $\{((0,1),(0,2)),((0,0),(0,2))\}$, as all we ever do is repair component type 1, and "leapfrog" state $((1,0),(0,2))$. We also have recurrent class $\{((0,1),(1,0)),((0,1),(1,1)),((0,1),(0,1)),((0,1),(0,0)),((0,1),(2,0)))\}$. This is because when starting from any of these states, we follow a "fully active" maintenance policy on the two components of type 2, whilst ignoring component 1. From this class, we never degrade into state $((0,1),(0,2))$, as degradations happen one at a time, and are always immediately placed under repair. Therefore, the chain induced by this policy has two recurrent classes, therefore DMP is not unichain for this configuration of $N$ and $M_i$. □

### A.1.2 Efficient Construction of IDDMP State Space

In Section 4.2.2 of the main body, we note that many of the states in $\mathcal{S}^{\max}$ and many of the state-action pairs are infeasible, that is to say that $\pi(\boldsymbol{s}, \boldsymbol{a})$ must be zero because no feasible design allows for them to take on any other value. As such, it is ideal to avoid generating such states and state-action pairs at all, allowing for larger problem instances to be solved. Note that the $\mathcal{S}_i$ (the single-type state spaces) are already constructed so as to follow the knapsack constraints.

We construct the reduced state space using a dynamic programming approach, starting by producing the state space the only uses two component type, then using this to construct the state space that only uses three component types, and so on. Let $\boldsymbol{x}(\boldsymbol{s})$ represent the minimal design that allows for state $\boldsymbol{s}$, i.e. the design that has $M_i - s_{i,j,2}$ copies of each component type $i$. Let $\mathcal{S}^{\max}_{1:n}$ be the space of partial states that only allows for the first $n \leq N$ component types. If $\boldsymbol{s}_{1:n} \in \mathcal{S}^{\max}_{1:n}$, $[\boldsymbol{x}(\boldsymbol{s}_{1:n})]_i$ returns 0 for $i > n$. We start by defining $\mathcal{S}^{\max}_{1:2} = \{s_{1:2} \in \mathcal{S}_1 \times \mathcal{S}_2 : A\boldsymbol{x}(\boldsymbol{s}_{1:2}) \leq \boldsymbol{b}\}$. This set is constructed computationally by enumerating all pairs $(\boldsymbol{s}_1, \boldsymbol{s}_2)$ and checking the knapsack conditions. We then use the recurrence $\mathcal{S}^{\max}_{1:n} = \left\{\boldsymbol{s}_{1:n} \in \mathcal{S}^{\max}_{1:(n-1)} \times \mathcal{S}_n : A\boldsymbol{x}(\boldsymbol{s}_{1:n}) \leq \boldsymbol{b}\right\}$. To construct $\mathcal{S}^{\max}_{1:3}$ up to $\mathcal{S}^{\max}_{1:N}$, the latter of which we call $\mathcal{S}^{\max}_{knap}$, which can be used in place of $\mathcal{S}^{\max}$ as it includes all states for which $\pi(\boldsymbol{s}, \boldsymbol{a})$ can feasibly take a non-zero action. This method allows for $\mathcal{S}^{\max}_{knap}$ to be enumerated without explicitly constructing $\mathcal{S}^{\max}$ first.

The action spaces $\mathcal{A}(\boldsymbol{s})$ can be constructed similarly. For any state $\boldsymbol{s}$, we start with $\mathcal{A}_i([\boldsymbol{s}]_i) = \{0, ..., s_{ij2}\}$. Then:

$$\mathcal{A}_{1:2}((\boldsymbol{s}_1, \boldsymbol{s}_2)) = \{\boldsymbol{a}_{1:2} \in \mathcal{A}_1(\boldsymbol{s}_1) \times \mathcal{A}_2(\boldsymbol{s}_2) : A\boldsymbol{x}((\boldsymbol{s}_1, \boldsymbol{s}_2) \oplus \boldsymbol{a}_{1:2}) \leq \boldsymbol{b}\},$$

via abuse of notation on the impulsive operator (recall this is used to give every state-action pair a "target" state to which it is effectively identical). Recursively:

$$\mathcal{A}_{1:n}((\boldsymbol{s}_1, ..., \boldsymbol{s}_n)) = \left\{\boldsymbol{a}_{1:n} \in \mathcal{A}_{1:(n-1)}((\boldsymbol{s}_1, ..., \boldsymbol{s}_{n-1})) \times \mathcal{A}_n(\boldsymbol{s}_n) : A\boldsymbol{x}((\boldsymbol{s}_1, ..., \boldsymbol{s}_n) \oplus \boldsymbol{a}_{1:n}) \leq \boldsymbol{b}\right\}.$$

$\mathcal{A}_{1:N}(\boldsymbol{s})$, aka $\mathcal{A}_{knap}(\boldsymbol{s})$ can then be used in place of $\mathcal{A}(\boldsymbol{s})$ for all $\boldsymbol{s} \in \mathcal{S}^{\max}_{knap}$. When constructing the MILP model for IDDMP, we then need only construct the variables $\pi(\boldsymbol{s}, \boldsymbol{a})$ such that $\boldsymbol{s} \in \mathcal{S}^{\max}_{knap}$ and $\boldsymbol{a} \in \mathcal{A}_{knap}(\boldsymbol{s})$, and only need to construct the MDP balance equation constraints for states in $\mathcal{S}^{\max}_{knap}$. This allows the exact MILP formulation to scale better.

## A.2 Section 3 Supplement

Section A.2.1 provides a tighter bound of $M_i$, the maximum number of copies of component type $i$, for $\varepsilon-\delta-$DOP. This can reduce the number of binary decision variables in the model, and also allows for solutions to be obtained when there are no knapsack constraints. To prove this bound, we also prove that the objective function of $\varepsilon-\delta-$DOP is supermodular. Section A.2.2 provides the algorithms used for the application of APP to BO-IDDMP.

### A.2.1 Tighter Bound for Number of Component Copies

We state the bound in Proposition A.2.1.

**Proposition A.2.1.** *The optimal number of copies of component $i$ for $\varepsilon-\delta-$DOP is no more than*

$$M_i = \min\left\{\max\left\{\left\lceil\left(\frac{1}{\ln q_i}\right)\ln\left\{-\frac{q_i r_i}{[(1+\delta)c_N - c_i]\ln q_i}\right\}\right\rceil, \left\lceil\frac{\varepsilon}{\log q_i}\right\rceil\right\}, \min_j\{b_j/a_{ji}\}\right\}.$$

In order to prove this, we make use of an alternative formulation of $\varepsilon-\delta-$DOP which uses general integer variables instead of binary variables, which we call $\varepsilon-\delta-$DOP-G, and a few additional definitions and lemmas. We start with the model:

$$\begin{aligned}(\varepsilon-\delta-\text{DOP-G}) \qquad \min_{\boldsymbol{x}} \ & \sum_{i=1}^{N} r_i q_i x_i + \sum_{i=1}^{N} c_i(1-q_i^{x_i})\prod_{k<i} q_k^{x_k} + (1+\delta)c_N\prod_{i=1}^{N} q_i^{x_i} \\ \text{s.t. } & A\boldsymbol{x} \le \boldsymbol{b} \\ & \sum_{i=1}^{N}\log(q_i)x_i \le \varepsilon \\ & x_i \in \mathbb{Z}_{\ge 0} \text{ for all } i \in [N]\end{aligned}$$

We now analyse single-component-type and multi-component-type variations of the problem to work towards a proof of Proposition A.2.1.

**Lemma A.2.1.** *The objective function of $\varepsilon-\delta-$DOP-G when allowing for only one component type $i$ in the solution is a convex function, with minimum:*

$$x_i^* \in \left\{ \left\lfloor \left(\frac{1}{\ln q_i}\right) \ln\left\{ -\frac{q_i r_i}{[(1+\delta)c_N - c_i]\ln q_i} \right\} \right\rfloor, \left\lceil \left(\frac{1}{\ln q_i}\right) \ln\left\{ -\frac{q_i r_i}{[(1+\delta)c_N - c_i]\ln q_i} \right\} \right\rceil \right\}.$$

*Proof.* Consider the continuous relaxation of the objective function of ($\varepsilon-\delta-$DOP-G) with only one $x_i$ non-zero for some component $i$:

$$g_i : \mathbb{R}^+ \to \mathbb{R},\ g_i(x) = q_i r_i x + c_i(1-q_i^x) + (1+\delta)c_N q_i^x.$$

We compute first and second derivatives:

$$\begin{aligned} g_i'(x) &= q_i r_i - (\ln q_i)c_i q_i^x + (1+\delta)c_N(\ln q_i)q_i^x, \\ &= q_i r_i + [(1+\delta)c_N - c_i](\ln q_i)q_i^x \\ g_i''(x) &= [(1+\delta)c_N - c_i](\ln q_i)^2 q_i^x \geq 0 \end{aligned}$$

As $g_i''(x) \geq 0$, $g$ is convex and therefore $g$ restricted to integers is also convex. By solving $g_i'(x^*) = 0$ we obtain a global minimum at:

$$x^* = \left(\frac{1}{\ln q_i}\right) \ln\left\{ -\frac{q_i r_i}{[(1+\delta)c_N - c_i]\ln q_i} \right\},$$

hence the integer minimum will be this value rounded up or down. □

**Corollary A.2.1.** *$g_i$ is increasing for $x > x^*$, with $x^*$ as defined in Lemma A.2.1.*

We now wish to extend this notation of convexity to the case where we allow for multiple copies of each component. As such, we must introduce the notations of set functions and supermodularity.

**Definition A.2.1** (Set function)**.** *Let $\mathcal{L}$ be some set. A function $f : 2^{\mathcal{L}} \to \mathbb{R}$ whose domain is the power set of $\mathcal{L}$ is called a (real) set function, and $\mathcal{L}$ is called the base set.*

For us, $\mathcal{L}$ represents our set of different components. We can alter our representation of $g$ using set functions. Let $\mathcal{L}' = \{(i, n) : i \in \mathcal{L}, n \in \mathbb{N}\}$. This represents an expanded version of our component set $\mathcal{L}$ with an infinite number of unique copies of each component. We then enforce the following partial ordering:

$$(i, m) \preceq (j, n) \text{ for all } i < j, i, j \in \mathcal{L}$$
$$(i, m) \preceq (i, n) \text{ for all } i \in \mathcal{L}, m < n.$$

We then define the set-function equivalent of $g$, $g^{SET}$, as:

$$g^{SET} : 2^{\mathcal{L}'} \to \mathbb{R}, g^{SET}(K) = \sum_{(i,m)\in K} q_i r_i + \sum_{(i,m)\in K} c_i p_i \prod_{(j,n)\in K,(j,n)\prec(i,m)} q_j + (1+\delta)c_N \prod_{(i,m)\in K} q_i.$$

We can map decision variables $\boldsymbol{x}$ to subsets of $\mathcal{L}'$ as follows:

$$\mathcal{K} : \mathcal{X} \to 2^{\mathcal{L}'}, \mathcal{K}(\boldsymbol{x}) = \{(i, m) \in \mathcal{L}' : n \leq x_i\}.$$

We then obtain the equality $g(\boldsymbol{x}) = g^{SET}(\mathcal{K}(\boldsymbol{x}))$.

**Definition A.2.2** (Marginal Set function)**.** *Let $\mathcal{L}$ be a base set and $f : 2^{\mathcal{L}} \to \mathbb{R}$ be a set function. Let $\mathcal{X} \subset \mathcal{L}$ be some strict subset of $\mathcal{L}$ and $c \in \mathcal{L}\backslash\mathcal{X}$ be some element not contained in $\mathcal{X}$. Then the function $f_X : \mathcal{L}\backslash\mathcal{X} \to \mathbb{R}$ satisfying $f_X(c) = f(\mathcal{X}\cup\{c\}) - f(\mathcal{X})$ is called the marginal set function of $f$ at $\mathcal{X}$, and is denoted $f(c|\mathcal{X})$.*

**Definition A.2.3** (Set Function Supermodularity)**.** *Let $\mathcal{L}$ be the base set of some set function $f : 2^{\mathcal{L}} \to \mathbb{R}$. $f$ is said to be a supermodular set function if and only if, for any subsets $\mathcal{X} \subset \mathcal{Y} \subset \mathcal{L}$, and any element $c \in \mathcal{L}\backslash\mathcal{Y}$, the following holds:*

$$f(c|\mathcal{X}) \leq f(c|\mathcal{Y}).$$

With our definitions laid out, we may state and prove an important property of our objective function.

**Lemma A.2.2.** *$g^{SET}$ is a supermodular set function.*

*Proof.* We shall suppress the $SET$ superscript for brevity. Recall :

$$g(\mathcal{K}) = \sum_{(i,m)\in\mathcal{K}} q_i r_i + \sum_{(i,m)\in\mathcal{K}} c_i p_i \prod_{(j,n)\in\mathcal{K},(j,n)\preceq(i,m)} q_j + (1+\delta)c_N \prod_{(i,m)\in\mathcal{K}} q_i.$$

Clearly, the first sum is a modular set function, so we need only show the supermodularity of the following:

$$f(\mathcal{K}) := \sum_{(i,m)\in\mathcal{K}} c_i p_i \prod_{(j,n)\in\mathcal{K},(j,n)\preceq(i,m)} q_j + (1+\delta)c_N \prod_{(i,m)\in\mathcal{K}} q_i,$$

as the positive linear combination of supermodular functions is supermodular.

To show supermodularity, for any subsets $\mathcal{X} \subset \mathcal{Y} \subset \mathcal{L}'$, and any component $(i,m) \in \mathcal{L}'\backslash\mathcal{Y}$, we must show that:

$$f(\mathcal{X} \cup \{(i,m)\}) - f(\mathcal{X}) \leq f(\mathcal{Y} \cup \{(i,m)\}) - f(\mathcal{Y}).$$

To simplify notation, we may use $\mathcal{K}^{>(i,m)}$ and $\mathcal{K}^{<(i,m)}$ to denote the partitions of a set $\mathcal{K}$ with only the components greater than $(i,m)$ under the partial ordering, and the components considered lesser than $(i,m)$, respectively. We may also write:

$$f((i,m)|\mathcal{K}) := f(\mathcal{K} \cup \{(i,m)\}) - f(\mathcal{K}),$$

and $f(\mathcal{K};c)$ to be $f$ with $(1+\delta)c_N$ replaced by $c$, bounding the penalty to be the usage cost of some component, and ignoring all components with usage costs greater than this.

We first calculate $f(\mathcal{K} \cup \{(i,m)\})$ as follows:

$$
\begin{aligned}
f(\mathcal{K} \cup \{(i,m)\}) &= \sum_{(j,n)\in\mathcal{K}\cup\{(i,m)\}} \left( c_j p_j \prod_{(k,o)\in\mathcal{K}\cup\{(i,m)\},(k,o)<(j,n)} q_k \right) + (1+\delta) c_N \prod_{(j,n)\in\mathcal{K}\cup\{(i,m)\}} q_j \\
&= \sum_{(j,n)\in\mathcal{K},(j,n)<(i,m)} \left( c_j p_j \prod_{(k,o)\in\mathcal{K},(k,o)<(j,n)} q_k \right) \\
&\quad + c_i p_i \prod_{(j,n)<(i,m)} q_j + \sum_{(j,n)\in\mathcal{K},(j,n)>(i,m)} \left( c_j p_j \prod_{(k,o)\in\mathcal{K}\cup\{(i,m)\},(k,o)<(j,n)} q_k \right) \\
&\quad + (1+\delta) c_N \prod_{(j,n)\in\mathcal{K}\cup\{(i,m)\}} q_j \\
&= (p_i + q_i) \sum_{(j,n)\in\mathcal{K},(j,n)<(i,m)} \left( c_j q_j \prod_{(k,o)\in\mathcal{K},(k,o)<(j,n)} q_k \right) \\
&\quad + c_i p_i \prod_{(j,n)<(i,m)} q_j + q_i \sum_{(j,n)\in\mathcal{K},(j,n)>(i,m)} \left( c_j p_j \prod_{(k,o)\in\mathcal{K},(k,o)<(j,n)} q_k \right) \\
&\quad + q_i (1+\delta) c_N \prod_{(j,n)\in\mathcal{K}} q_j \\
&= c_i p_i \prod_{(j,n)<(i,m)} q_j \\
&\quad + p_i \sum_{(j,n)\in\mathcal{K},(j,n)<(i,m)} \left( c_j p_j \prod_{(k,o)\in\mathcal{K},(k,o)<(j,n)} q_k \right) \\
&\quad + q_i \left\{ \sum_{(j,n)\in\mathcal{K}} \left( c_j p_j \prod_{(k,o)\in\mathcal{K},(k,o)<(j,n)} q_k \right) + (1+\delta) c_N \prod_{(j,n)\in\mathcal{K}} q_j \right\} \\
&= p_i \left\{ c_i \prod_{(j,n)<(i,m)} q_j + \sum_{(j,n)\in\mathcal{K},(j,n)<(i,m)} \left( c_j p_j \prod_{(k,o)\in\mathcal{K},(k,o)<(j,n)} q_k \right) \right\} \\
&\quad + q_i f(\mathcal{K}) \\
&= p_i f(\mathcal{K}^{<(i,m)}; c_i) + q_i f(\mathcal{K}).
\end{aligned}
$$

This makes intuitive sense. By including an extra component $i$, with probability $p_i$ you are at worst using component $i$ and ignoring all components worse than $i$, and with

probability $q_i$ the link is unavailable and the system falls back onto the original links in $\mathcal{K}$. Notably, this shows that $f$ is monotone. We then see that:

$$\begin{aligned} f((i,m)|\mathcal{K}) &= f(\mathcal{K} \cup \{(i,m)\}) - f(\mathcal{K}) \\ &= p_i \left( f(\mathcal{K}^{<(i,m)}; c_i) - f(\mathcal{K}) \right). \end{aligned}$$

As the constant $p_i$ does not depend on $\mathcal{K}$, we need only focus on $f(\mathcal{K}^{<(i,m)}; c_i) - f(\mathcal{K})$. We see:

$$\begin{aligned} f(\mathcal{K}^{<(i,m)}; c_i) - f(\mathcal{K}) &= c_i \prod_{(j,n)\in\mathcal{K},(j,n)<(i,m)} q_j + \sum_{(j,n)\in\mathcal{K},(j,n)<(i,m)} \left( c_j p_j \prod_{(k,o)\in\mathcal{K},(k,o)<(j,n)} q_k \right) \\ &\quad - \left\{ \sum_{(j,n)\in\mathcal{K}} \left( c_j p_j \prod_{(k,o)\in\mathcal{K},(k,o)<(j,n)} q_k \right) + (1+\delta) c_N \prod_{(j,n)\in\mathcal{K}} q_j \right\} \\ &= c_i \prod_{(j,n)\in\mathcal{K},(j,n)<(i,m)} q_j - \sum_{(j,n)\in\mathcal{K},(j,n)>(i,m)} \left( c_j p_j \prod_{(k,o)\in\mathcal{K},(k,o)<(j,n)} q_k \right) \\ &\quad - (1+\delta) c_N \prod_{(j,n)\in\mathcal{K}} q_j \\ &= \left\{ \prod_{(j,n)\in\mathcal{K},(j,n)<(i,m)} q_j \right\} \\ &\quad \times \left\{ c_i - \sum_{(j,n)\in\mathcal{K},(j,n)>(i,m)} c_j p_j \prod_{(k,o)\in\mathcal{K},(i,m)<(k,o)<(j,n)} q_k \right. \\ &\qquad \left. -(1+\delta) c_N \prod_{(j,n)\in\mathcal{K},(j,n)>(i,m)} q_j \right\} \\ &= \left\{ \prod_{(j,n)\in\mathcal{K},(j,n)<(i,m)} q_j \right\} \left\{ c_i - f(\mathcal{K}^{>(i,m)}) \right\} \\ &= - \left\{ \prod_{(j,n)\in\mathcal{K},(j,n)<(i,m)} q_j \right\} \left\{ f(\mathcal{K}^{>(i,m)}) - c_i \right\}. \end{aligned}$$

The magnitude of this expression decreases as we add more links to $\mathcal{K}$ that are

cheaper than $i$ by the first term, and the magnitude also decreases as we add more links more expensive than $i$ by the second term, as $f$ is monotone decreasing and bounded below by $c_i$. However, the expression is negative and therefore increasing in $\mathcal{K}$, therefore the function is supermodular. □

Using these properties, we may now prove Proposition A.2.1.

*Proof.* Firstly, assume we have no knapsack constraints. We assume that our bound must allow us to represent the following feasible solution:

$$\mathcal{K} = \left\{ (i,n) \,|\, n \leq \max\left\{ \left\lceil \left( \frac{1}{\ln q_i} \right) \ln \left\{ - \frac{q_i r_i}{[(1+\delta)c_N - c_i] \ln q_i} \right\} \right\rceil, \left\lceil \frac{\varepsilon}{\ln q_i} \right\rceil \right\} \right\}.$$

This solution is feasible according to the $\varepsilon$-constraint, and cannot be improved by adding more copies of component $i$ by Lemma A.2.1. Extending this, for any sets $\mathcal{K}' \supset \mathcal{K}$, we have $g^{SET}(\mathcal{K}' \cup \{(i, |\mathcal{K}|)\}) \geq g^{SET}(\mathcal{K}')$ by supermodularity. In other words, if adding an extra copy of component $i$ to $\mathcal{K}$ would worsen the solution, then adding an extra copy of $i$ to any solution containing $\mathcal{K}$ would also worsen that solution.. As such, we need not consider solutions with more than $|\mathcal{K}|$ copies of component $i$, so we obtain bound $M_i^{NO-KNAP} = \max\left\{ \left\lceil - \frac{q_i r_i}{[(1+\delta)c_N - c_i] \ln q_i} \right\rceil, \left\lceil \frac{\varepsilon}{\ln q_i} \right\rceil \right\}$. Now adding back in knapsack constraints, we need not consider solutions that violate any knapsack constraint, so we strengthen our bound to $M_i = \min\{M_i^{NO-KNAP}, \min_j\{b_j / a_{ji}\}\}$, as desired. □

### A.2.2 Algorithms

Here, we provide the algorithms used to implement the APP methodology for BO-IDDMP.

Algorithm 8 takes as input a starting value $\varepsilon_{\min}$, an increment value $\Delta\varepsilon$, and the problem parameters, and returns a set of Pareto-optimal solutions to this problem. It begins by obtaining the two objective values and design solution for F-DOP, giving a lower-bound on $\varepsilon$ and providing the most extreme solution in terms of reliability. Then starting with $\varepsilon = \varepsilon_{\min}$, we iteratively solve $\varepsilon - \delta-$DOP-PC, store the resulting solution $(g^o, \ln g^f, x)$, and update $\varepsilon$ to $\ln g^f + \Delta\varepsilon$ , ensuring that the next solution is more reliable. We do this until the lower bound on $\epsilon$ is reached, at which point we return all objective values and solutions found.

**Algorithm 8** SP1

**Require:** $\varepsilon_{\min}, \Delta\varepsilon, \mathcal{L} = (N, \alpha, \tau, c, r, A, b)$
  StaticSolutions $\leftarrow$ [ ]
  $(g_f^o, \ln g_f^f, x_f) \leftarrow$ F-DOP($\mathcal{L}$)
  StaticSolutions $\leftarrow [(g_f^o, \ln g_f^f, x_f)]$
  $\varepsilon \leftarrow \varepsilon_{\min}$
  **while** $\varepsilon \leq \ln g_f^f$ **do**
    $(g^o, \ln g^f, x) \leftarrow \varepsilon - \delta -$ DOP-PC($\mathcal{L}, \varepsilon$)
    push(StaticSolutions, $(g^o, \ln g^f, x)$)
    $\varepsilon \leftarrow \ln g^f + \Delta\varepsilon$
  **end while**
    **return** StaticSolutions

Algorithm 9 takes as input a design $x$, a label for that design $i$, a minimum LFR $\ln g_{static}^f$, a minimum penalty $p_{\min}$, a multiplicative increment $\Delta p$, and the problem parameters. It returns a set of Pareto-optimal solutions. Starting with $p = p_{\min}$, it solves $p-$DMP for the design $x$ and stores the tuple of the objective values, design label, and penalty used, $(g^o, \ln g^f, i, p)$. It then performs the update $p \leftarrow p \times \Delta p$, and loops back around. It does this until $\ln g^f \neq \ln g_{static}^f$, at which point the penalty $p$ has been increased far enough that the fully-active maintenance policy has been recovered.

It then returns all solutions found.

**Algorithm 9** SP2

**Require:** $x, i, \ln g^f_{static}, p_{\min}, \Delta p, \mathcal{L} = (N, \alpha, \tau, c, r, A, b)$
  DynamicSolutions $\leftarrow [\,]$
  $p \leftarrow p_{\min}$
  $(g^o, g^f) \leftarrow (0, 0)$
  **while** $\ln g^f \neq \ln g^f_{static}$ **do**
    $(g^o, g^f) \leftarrow p$-DMP$(\mathcal{L}, x, p)$
    push(DynamicSolutions, $(g^o, \ln g^f, i, p)$)
    $p \leftarrow p \times \Delta p$
  **end while**
    **return** DynamicSolutions

Algorithm 10 is the full algorithm for applying APP to BO-IDDMP. It takes as input $\varepsilon_{\min}, \Delta\varepsilon, p_{\min}, \Delta p, \delta$, and the problem parameters. It constructs an approximate set of Pareto-optimal solutions to the problem. Alongside the separate functions for SP1 and SP2$(x)$, it makes use of two helper functions: NonNestedDesigns and NonDomSolutions. NonNestedDesigns takes as input a list of three-tuples where the first two entries contain objective values, and the third entry contains decision variables representing a design. A design $x_1$ is considered "nested" in another design $x_2$ if the $x_2$ contains at least as many copies of each type of component as $x_1$, and at least one extra component. The purpose of eliminating such designs is that any dynamic policy on design $x_1$ can be emulated on $x_2$ by simply ignoring the extra components. As such, obtaining a Pareto-front of policies for each design would result in repeated solutions and wasted computational time. NonNestedSolutions therefore checks all designs against all other designs to check for nesting, and returns the sublist that only contains solutions which are not nested in other solutions. NonDomSolutions takes a list of tuples of varying sizes where the first two components of the tuple describes the two objectives. It iterates over all solutions and checks them against all other solutions to check for domination of one solution over another. It then returns the sublist of non-dominated tuples.

Both NonNestedDesigns and NonDomSolutions use very similar logic to check for

**Algorithm 10** APP

**Require:** $\varepsilon_{\min}, \Delta\varepsilon, p_{\min}, \Delta p, \delta, \mathcal{L} = (N, \alpha, \tau, c, r, A, b)$
  StaticSolutions$\leftarrow$ SP1$(\varepsilon_{\min}, \Delta\varepsilon, \mathcal{L})$
  NonNestedDesigns $\leftarrow$NonNestedDesigns(StaticSolutions)
  DynamicSolutions $\leftarrow$ [ ]
  **for** $i = 1, ...,$length(NonNestedDesigns) **do**
    $(g^o_{static}, \ln g^f_{static}, x) \leftarrow$ NonNestedDesigns$[i]$
    NewDynamicSolutions$\leftarrow$ SP2$(x, i, \ln g^f_{static}, p_{\min}, \Delta p, \delta, \mathcal{L})$
    append(DynamicSolutions, NewDynamicSolutions)
  **end for**
  Population $\leftarrow$ (StaticSolutions$\backslash$NonNestedDesigns) $\cup$ DynamicSolutions
  NonDomSolutions $\leftarrow$ NonDomSolutions(Population)

domination or nesting, so for the sake of brevity we only show the algorithm for NonDomSolutions in Algorithm 11.

**Algorithm 11** NonDomSolutions

**Require:** Population
  **for** sol $\in$ Population **do**
    dominated $\leftarrow$ False
    **for** sol$'$ $\in$ Population$\backslash${sol} **do**
      **if** sol$'[1] \leq$ sol[1] and sol$'[2] \leq$ sol[2] and (sol$'[1] <$ sol[1] or sol$'[2] <$ sol[2]) **then**
        dominated $\leftarrow$ True
        Break
      **end if**
    **end for**
    **if** not dominated **then**
      push(NonDomSolutions, sol)
    **end if**
  **end for**

## A.3 Section 4 Supplement

### A.3.1 Results Table for Section 4.2

In the Heuristic solutions column, $(-x)$ indicates that $x$ of the solutions found were dominated by an exact solution. In the DOP solutions column, $(-x)$ indicates that $x$

| Problem Instance | | LP | | Heuristic | | DOP | |
|---|---|---|---|---|---|---|---|
| Comp. Set | Budget | Time | Solutions | Time | Solutions | Time | Solutions |
| 1 | 6 | 0.38 | 6 | 0.95 | 8 | 0.011 | 4 |
| 1 | 8 | 0.86 | 9 | 0.99 | 15 | 0.032 | 5 |
| 1 | 10 | 3.09 | 14 | 1.3 | 24 | 0.024 | 6 |
| 1 | 12 | 14.46 | 22 | 1.37 | 34 | 0.038 | 7 |
| 1 | 14 | 63.84 | 31 | 1.81 | 54 | 0.049 | 8(-1) |
| 1 | 16 | 294.75 | 38 | 2.48 | 86 | 0.075 | 9(-1) |
| 2 | 24 | 0.38 | 8 | 0.32 | 7 | 0.02 | 4 |
| 2 | 32 | 0.51 | 12 | 0.33 | 11 | 0.028 | 5 |
| 2 | 40 | 1.72 | 19 | 0.34 | 17 | 0.034 | 6 |
| 2 | 48 | 6.47 | 27 | 0.35 | 26 | 0.023 | 7 |
| 2 | 56 | 26.82 | 36 | 0.36 | 32 | 0.026 | 8(-3) |
| 2 | 64 | 134.35 | 43 | 0.41 | 41 | 0.042 | 9(-4) |
| 3 | 12 | 0.34 | 8 | 0.33 | 8 | 0.008 | 4 |
| 3 | 16 | 0.53 | 13 | 0.34 | 11 | 0.011 | 5 |
| 3 | 20 | 1.63 | 17 | 0.36 | 19 | 0.02 | 6 |
| 3 | 24 | 6.23 | 26 | 0.37 | 24 | 0.027 | 7(-2) |
| 3 | 28 | 34.84 | 34 | 0.41 | 34 | 0.053 | 8(-3) |
| 3 | 32 | 165.31 | 44 | 0.42 | 41 | 0.047 | 9(-4) |
| 4 | 12 | 0.31 | 4 | 0.32 | 4 | 0.011 | 3 |
| 4 | 16 | 0.41 | 8 | 0.33 | 7 | 0.014 | 4 |
| 4 | 20 | 0.87 | 12 | 0.32 | 11 | 0.012 | 5 |
| 4 | 24 | 4.76 | 30 | 0.38 | 31 | 0.022 | 6 |
| 4 | 28 | 57.69 | 45 | 0.9 | 68 | 0.038 | 7(-1) |
| 4 | 32 | 207.76 | 42 | 1.57 | 93 | 0.054 | 8(-1) |
| 5 | 9 | 0.35 | 7 | 0.64 | 7 | 0.01 | 4 |
| 5 | 12 | 1.05 | 12 | 0.95 | 10(-1) | 0.013 | 5 |
| 5 | 15 | 2.34 | 14 | 1.01 | 36 | 0.018 | 6 |
| 5 | 18 | 14.56 | 27 | 1.27 | 52 | 0.044 | 7 |
| 5 | 21 | 97.07 | 39 | 2.17 | 65 | 0.035 | 8 |
| 5 | 24 | 382.46 | 54 | 5.44 | 109(-6) | 0.226 | 9 |
| 6 | 12 | 0.37 | 5 | 0.64 | 6 | 0.009 | 4 |
| 6 | 16 | 0.81 | 9 | 0.64 | 9 | 0.017 | 5 |
| 6 | 20 | 4.08 | 14 | 0.68 | 12 | 0.033 | 6 |
| 6 | 24 | 66.83 | 39 | 0.73 | 32 | 0.025 | 7 |
| 6 | 28 | 4624.9 | 31 | 1.26 | 60 | 0.044 | 8 |
| 6 | 32 | 7196.05 | 0 | 1.87 | 62 | 0.054 | 9(-3) |
| 7 | 21 | 0.33 | 5 | 0.63 | 6 | 0.007 | 4 |
| 7 | 28 | 0.45 | 9 | 0.64 | 10 | 0.009 | 5 |
| 7 | 35 | 1.88 | 20 | 0.67 | 22 | 0.014 | 6 |
| 7 | 42 | 32.8 | 39 | 1.23 | 63 | 0.026 | 7 |
| 7 | 49 | 145.97 | 39 | 2.33 | 83 | 0.043 | 8 |
| 7 | 56 | 385.03 | 31 | 6.3 | 83 | 0.078 | 9 |

Table A.3.1: Comparison of runtimes and numbers of solutions found for each method. (Part I)

| Problem Instance | | LP | | Heuristic | | DOP | |
|---|---|---|---|---|---|---|---|
| Comp. Set | Budget | Time | Solutions | Time | Solutions | Time | Solutions |
| 8 | 12 | 0.32 | 5 | 0.66 | 5 | 0.008 | 4 |
| 8 | 16 | 0.42 | 10 | 0.67 | 10 | 0.009 | 5 |
| 8 | 20 | 1.15 | 11 | 1.08 | 19 | 0.019 | 6 |
| 8 | 24 | 2.41 | 15 | 1.09 | 24 | 0.02 | 7 |
| 8 | 28 | 12.97 | 31 | 1.33 | 42 | 0.027 | 8 |
| 8 | 32 | 77.89 | 36 | 2.35 | 78 | 0.06 | 9(-1) |
| 9 | 21 | 0.36 | 5 | 0.64 | 5 | 0.011 | 4 |
| 9 | 28 | 0.57 | 8 | 0.33 | 8 | 0.01 | 4 |
| 9 | 35 | 4.98 | 20 | 0.37 | 19 | 0.016 | 5 |
| 9 | 42 | 254.48 | 40 | 0.89 | 51 | 0.04 | 6 |
| 9 | 49 | 2938.78 | 36 | 1.92 | 60 | 0.035 | 7 |
| 9 | 56 | 7196.93 | 0 | 0.39 | 22 | 0.055 | 7(-1) |
| 10 | 15 | 0.34 | 5 | 0.64 | 6 | 0.007 | 4 |
| 10 | 20 | 0.46 | 9 | 0.65 | 9 | 0.01 | 5 |
| 10 | 25 | 1.23 | 14 | 0.65 | 13 | 0.032 | 6 |
| 10 | 30 | 4.79 | 19 | 0.35 | 17 | 0.02 | 6 |
| 10 | 35 | 39.9 | 46 | 0.47 | 49 | 0.03 | 7(-2) |
| 10 | 40 | 916.98 | 88 | 1.28 | 113 | 0.034 | 8(-3) |
| 11 | 15 | 0.34 | 5 | 0.64 | 6 | 0.008 | 4 |
| 11 | 20 | 0.45 | 9 | 0.66 | 9 | 0.01 | 5 |
| 11 | 25 | 1.29 | 14 | 0.68 | 15 | 0.019 | 6 |
| 11 | 30 | 5.44 | 20 | 0.69 | 18 | 0.022 | 7 |
| 11 | 35 | 56.36 | 46 | 0.81 | 49 | 0.032 | 8 |
| 11 | 40 | 1160.02 | 79 | 1.74 | 106 | 0.054 | 9(-1) |
| 12 | 12 | 0.33 | 5 | 0.69 | 6 | 0.01 | 4 |
| 12 | 16 | 0.77 | 7 | 1.16 | 10 | 0.027 | 5 |
| 12 | 20 | 2.8 | 12 | 1.25 | 19 | 0.029 | 6 |
| 12 | 24 | 23.32 | 15 | 1.71 | 38 | 0.096 | 7 |
| 12 | 28 | 94.76 | 17 | 2.97 | 37 | 0.39 | 8 |
| 12 | 32 | 524.15 | 33 | 5.53 | 61 | 0.674 | 9 |
| 13 | 15 | 0.38 | 8 | 0.4 | 8 | 0.009 | 4 |
| 13 | 20 | 0.76 | 14 | 0.35 | 11 | 0.012 | 5 |
| 13 | 25 | 2.27 | 20 | 0.36 | 17 | 0.03 | 6 |
| 13 | 30 | 7.99 | 21 | 0.41 | 22 | 0.03 | 7(-1) |
| 13 | 35 | 79.29 | 35 | 0.46 | 30 | 0.037 | 8(-1) |
| 13 | 40 | 615.59 | 40 | 0.44 | 36 | 0.043 | 9(-1) |
| 14 | 18 | 0.35 | 4 | 0.32 | 4 | 0.007 | 3 |
| 14 | 24 | 0.94 | 10 | 0.32 | 11 | 0.011 | 4 |
| 14 | 30 | 6.0 | 13 | 0.64 | 20 | 0.018 | 5 |
| 14 | 36 | 117.9 | 18 | 0.32 | 11 | 0.026 | 5 |
| 14 | 42 | 3410.58 | 34 | 0.37 | 29 | 0.028 | 6 |
| 14 | 48 | 5974.65 | 6 | 0.86 | 57 | 0.05 | 7 |

Table A.3.2: Comparison of runtimes and numbers of solutions found for each method. (Part II)

solutions were dominated by an APP solution, demonstrating where a dynamic policies not only provided a more complete Pareto front, but also dominated an always-maintained design solution across both objectives.

### A.3.2 Additional Plots for Section 4.2

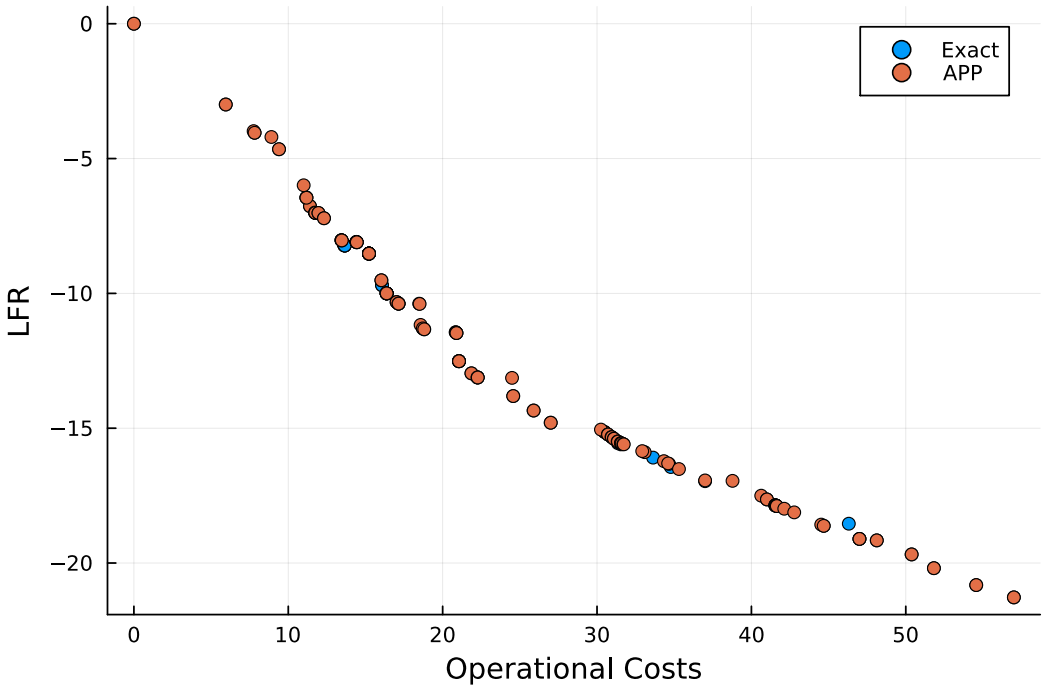



Figure A.3.1: Pareto front for instance (5,24)

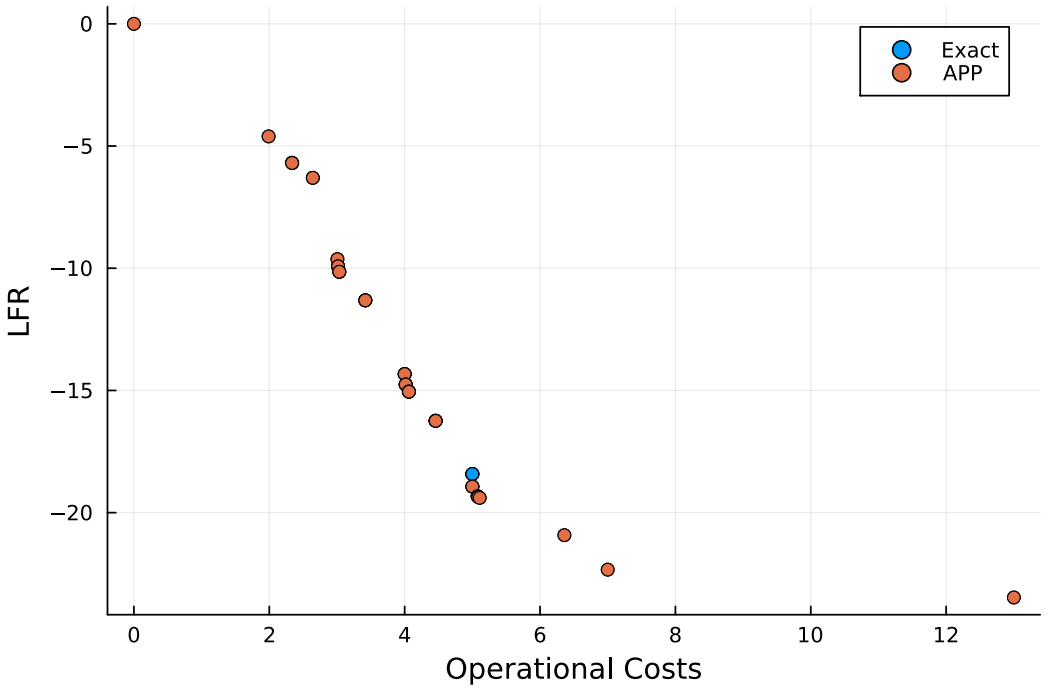



Figure A.3.2: Pareto front for instance (6,24)

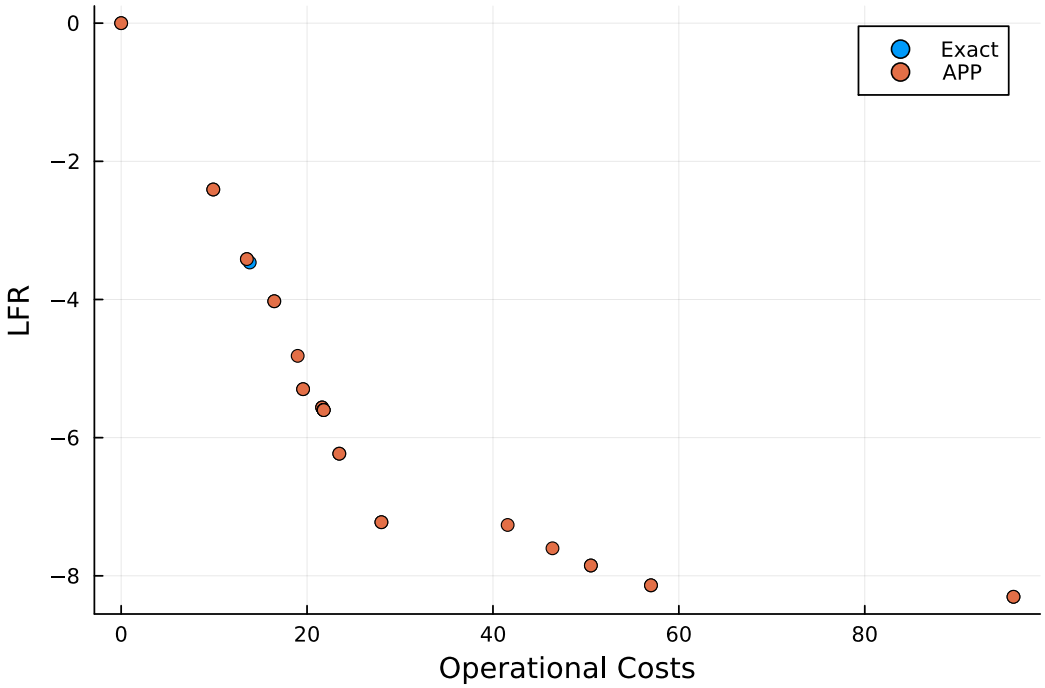



Figure A.3.3: Pareto front for instance (8,20)

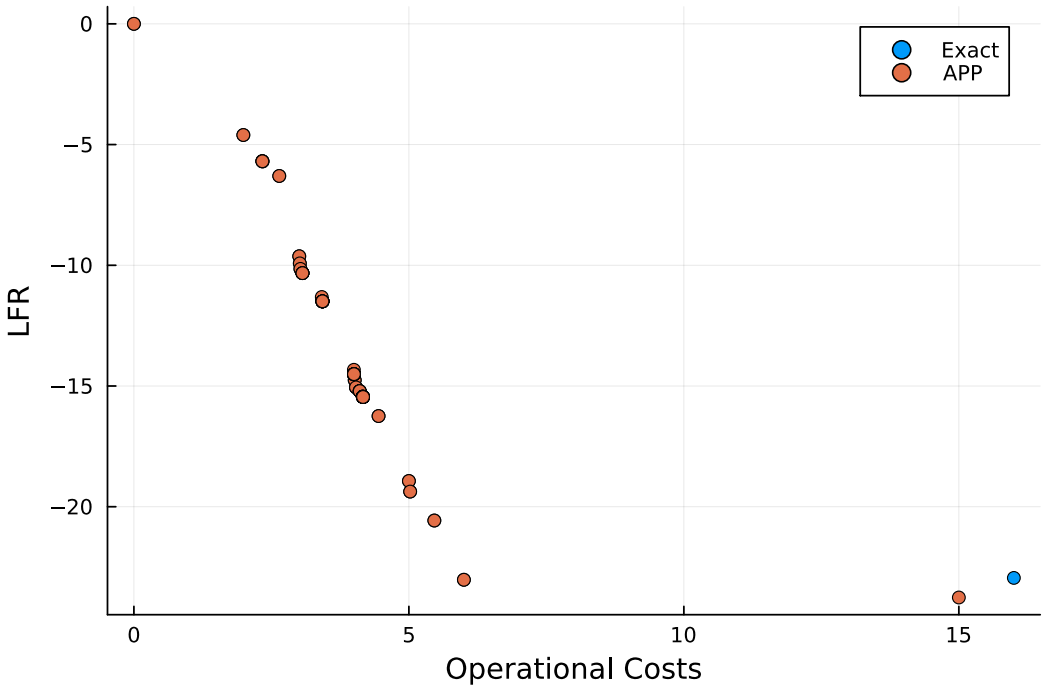



Figure A.3.4: Pareto front for instance (9,49)

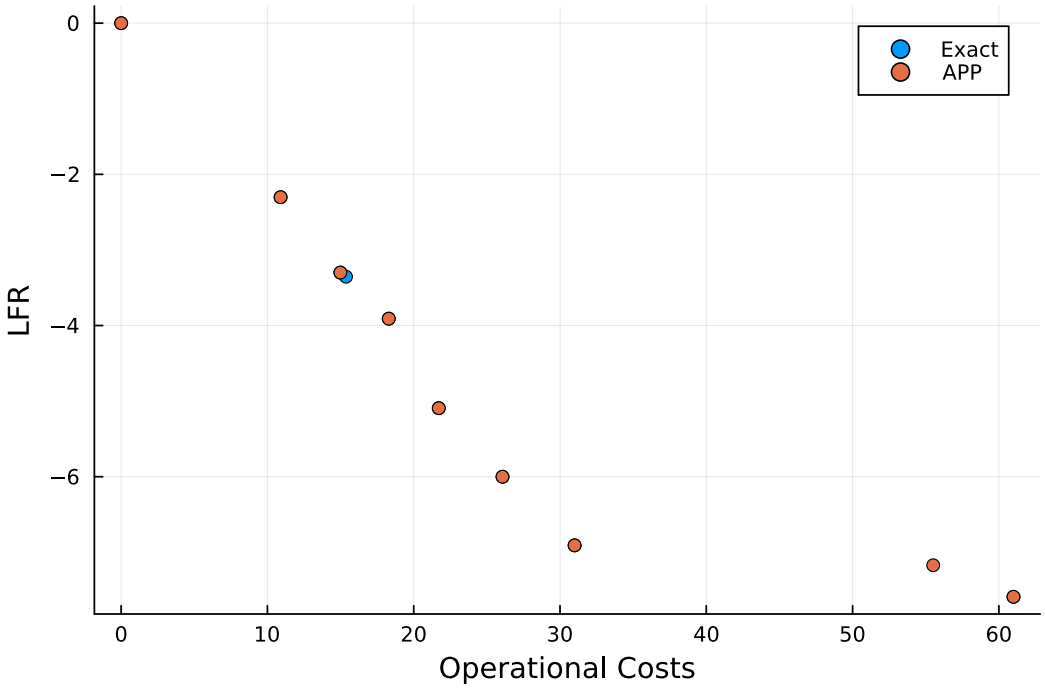



Figure A.3.5: Pareto front for instance (10,20)

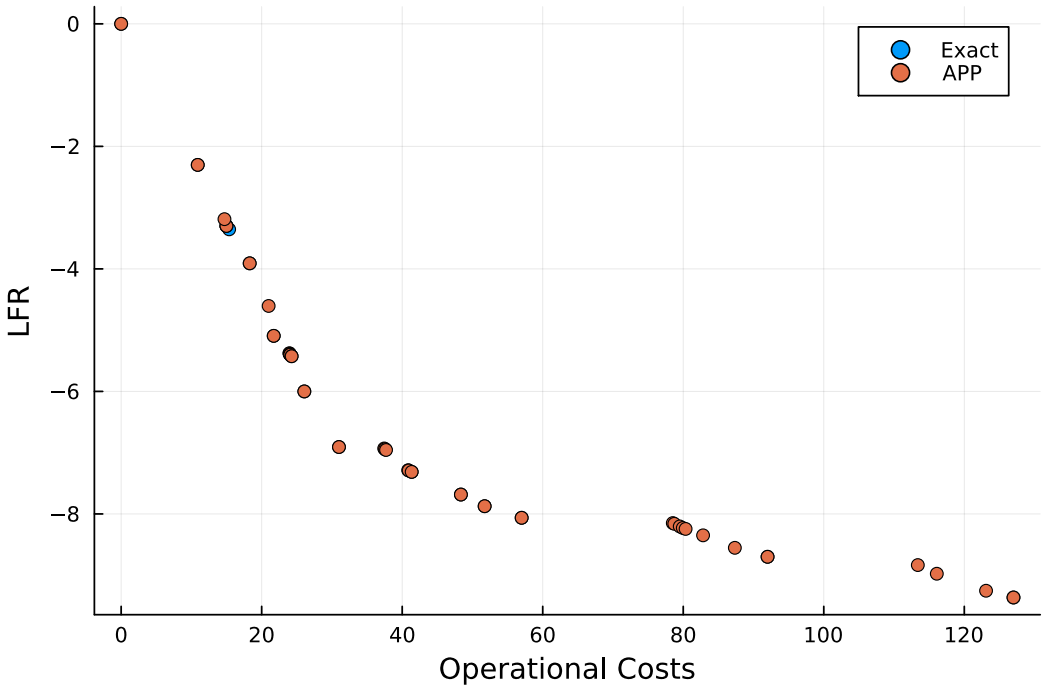



Figure A.3.6: Pareto front for instance (12,24)

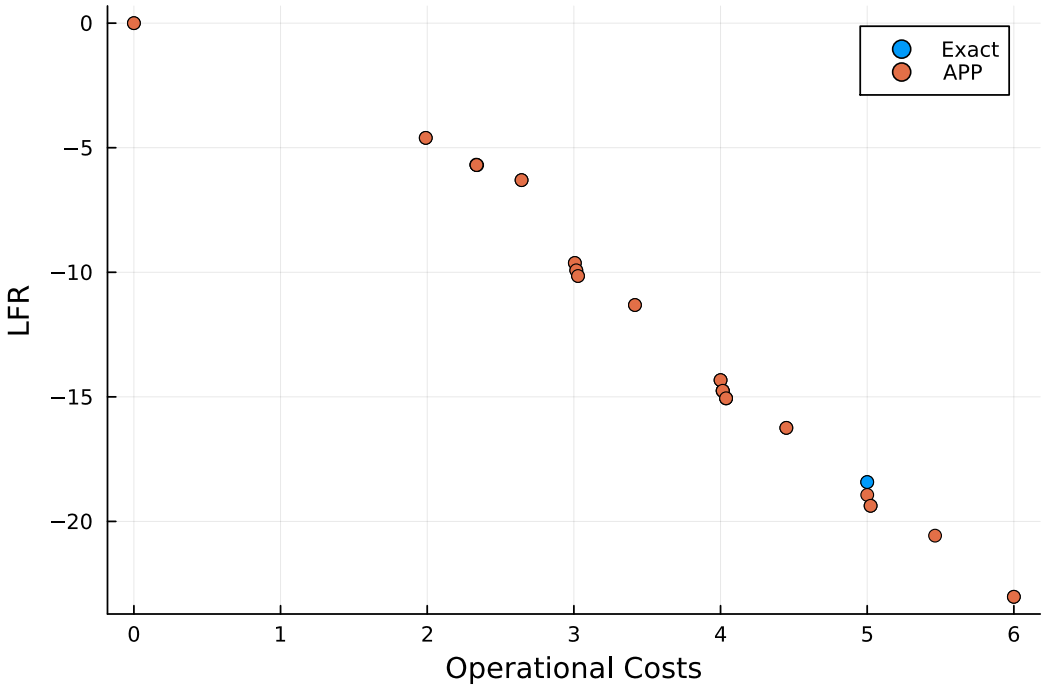



Figure A.3.7: Pareto front for instance (13,25)

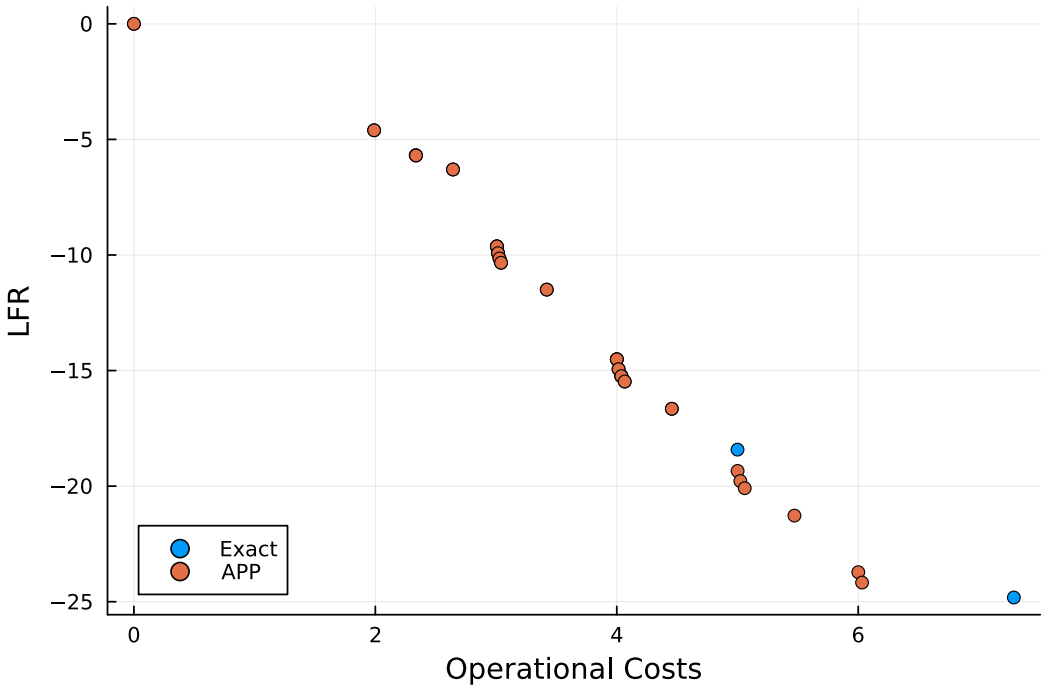



Figure A.3.8: Pareto front for instance (13,30)

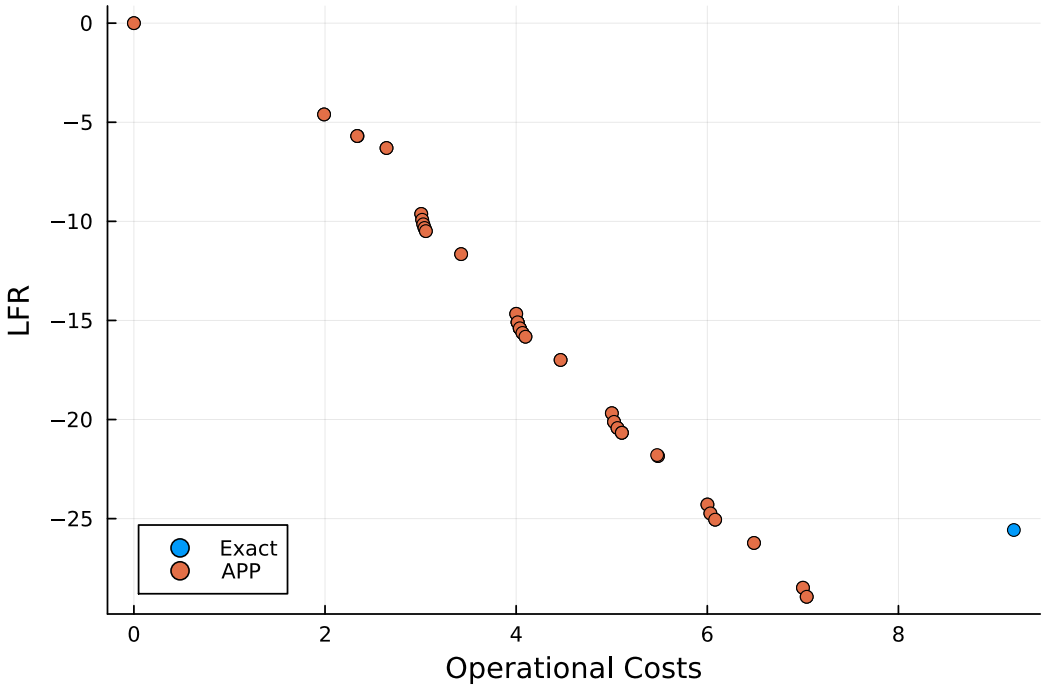



Figure A.3.9: Pareto front for instance (13,35)

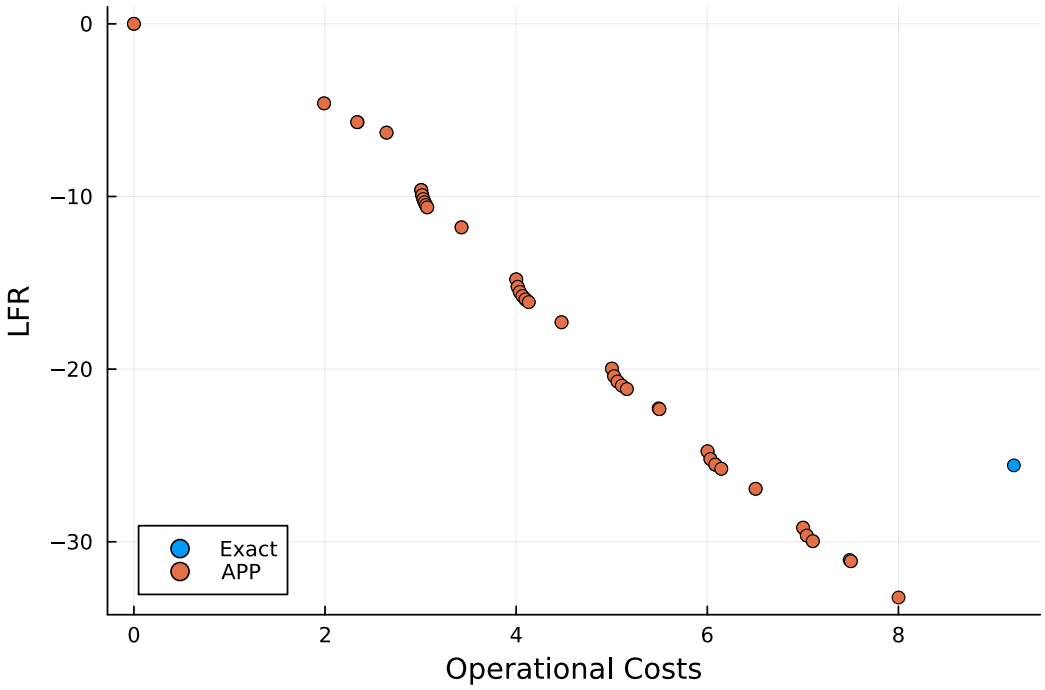



Figure A.3.10: Pareto front for instance (13,40)

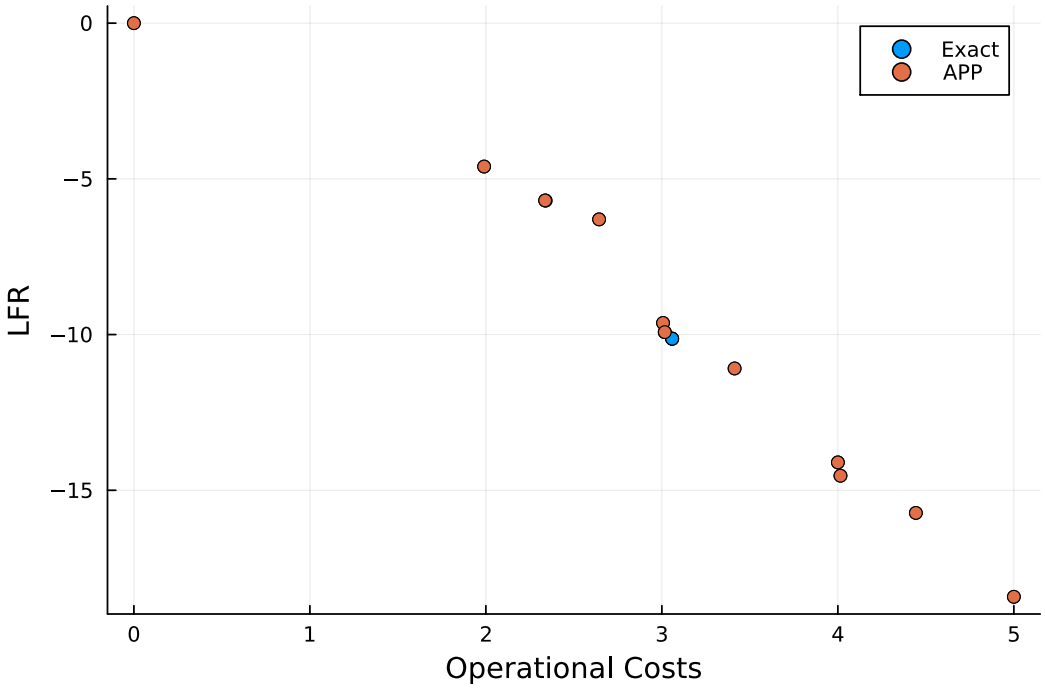



Figure A.3.11: Pareto front for instance (14,36)

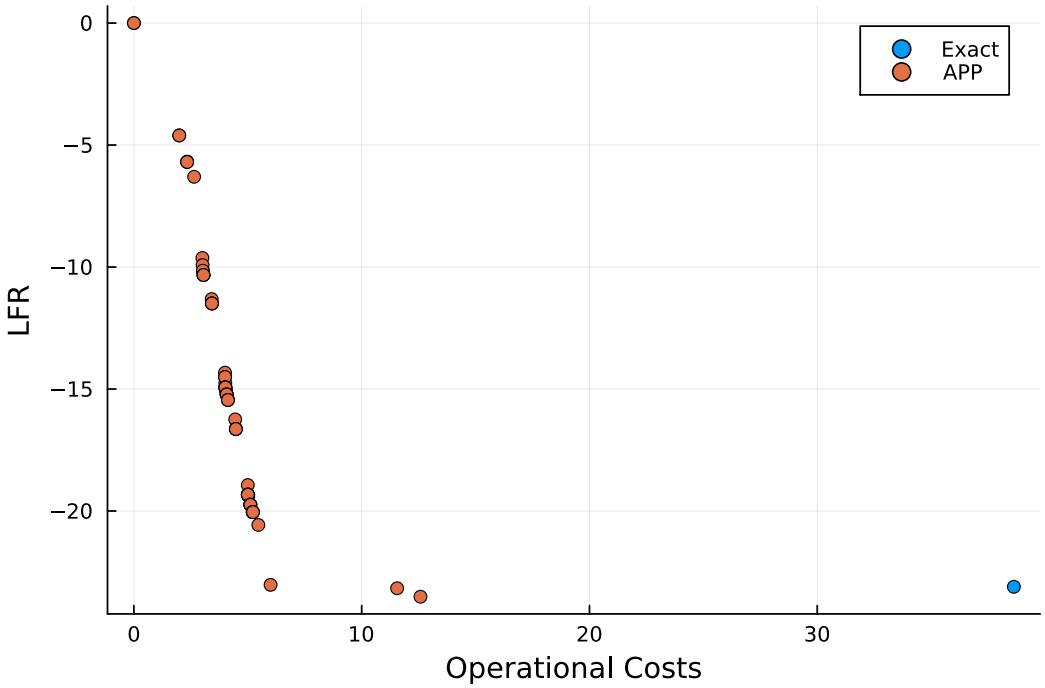


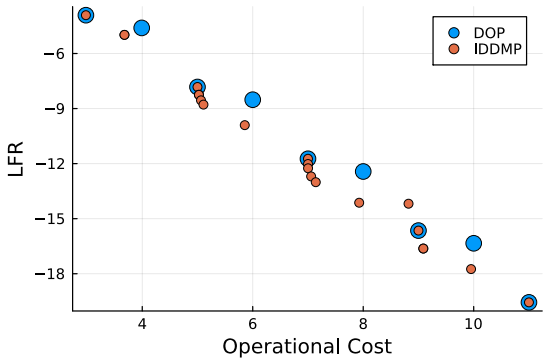

Figure A.3.12: Pareto front for instance (14,48)

### A.3.3 Effect of event rate scaling

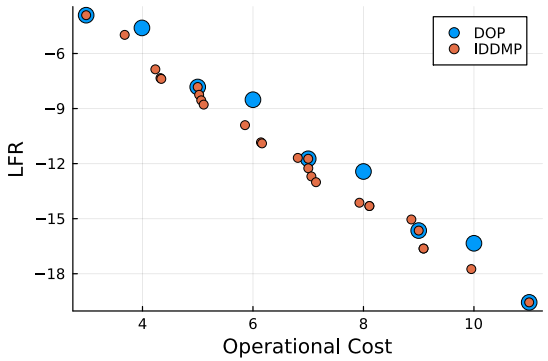
In this subsection we explore the effects of 'event rate scaling', which refers to the simultaneous scaling of the $\alpha_i$ and $\tau_i$ parameters of individual component types. We again perform our analysis on instance 6-20, and further explore the $(r_1, r_2) = (300, 100)$ case. If we multiply the values $\alpha_i$ and $\tau_i$ by some constant factor $m_i$, the reliability $p_i$ remains the same and therefore the underlying designs found by BO-DOP remain the same; however, the dynamic policies may yield different results. The effect of this multiplier is twofold: both failure and repair events happen faster, and the expected total cost of a single repair is affected, as this takes the value $r_i/(m_i\tau_i)$.

We consider multiplier values of 1.0, 5.0, and 10.0, and all combinations thereof, applied to components 1 and 2. As in the previous subsection, components 3 and 4 are ignored for this problem. Figure A.3.13 shows the Pareto fronts for all combinations of the multipliers applied to components 1 and 2. It is immediately apparent that these fronts are not all the same, so the simultaneous scaling of these rates does indeed

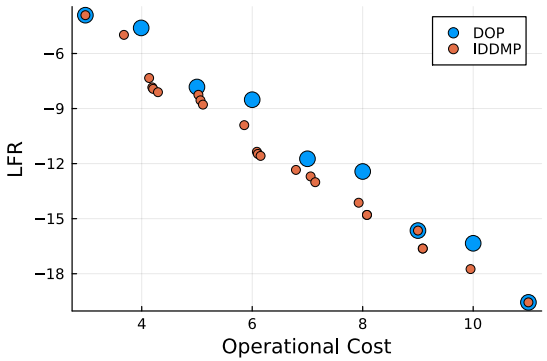
Figure A.3.13: Comparison of Pareto fronts when varying event rates.



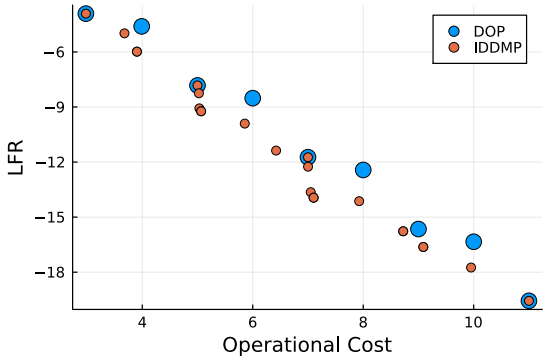

(a) $1.0, 1.0$



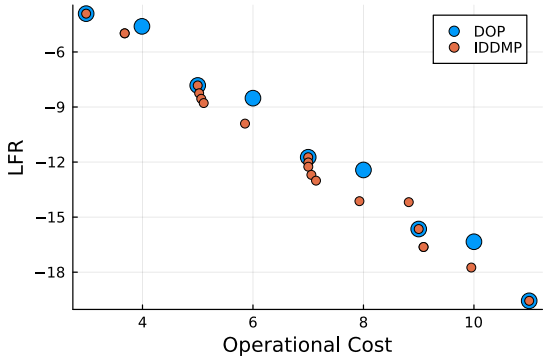

(b) $1.0, 5.0$



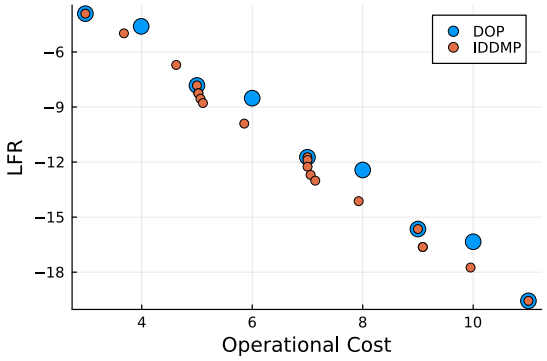

(c) $1.0, 10.0$



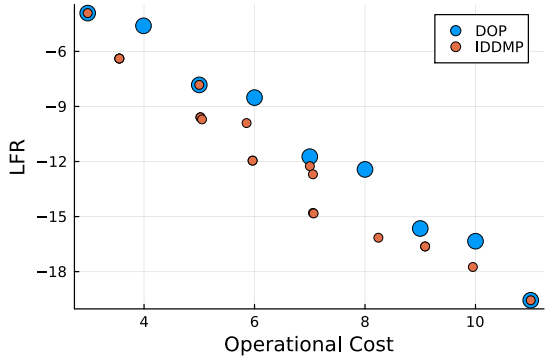

(d) $5.0, 1.0$



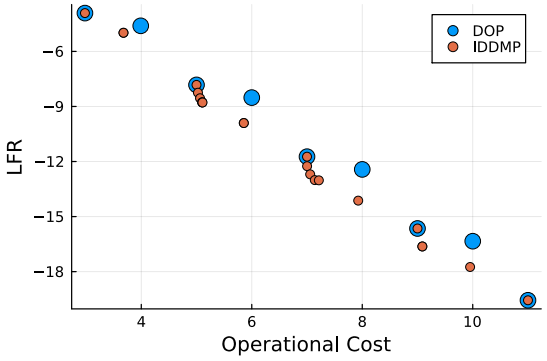

(e) $5.0, 5.0$



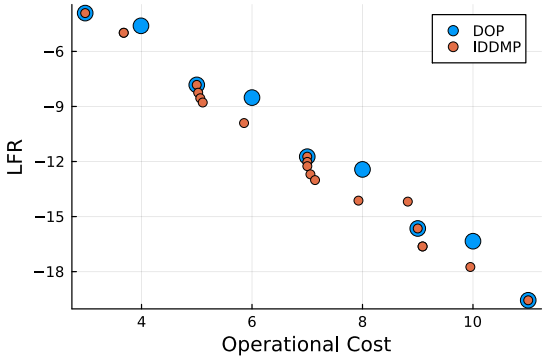

(f) $5.0, 10.0$




(g) $10.0, 1.0$




(h) $10.0, 5.0$




(i) $10.0, 10.0$

affect the outcomes of dynamic policies, despite not affecting the reliability value $p_i$. As previously mentioned, the design solutions are unaffected by the scaling of the rates, so the DOP solutions are the same across all cases. The fronts are identical for combinations (1,1), (5,5), and (10,10). This is because the ratio of the multipliers is equivalent across these cases, so these multipliers simply correspond to the re-scaling of time, and hence the solutions are unaffected. We also note that the fronts for combinations (5,10) and (10,5) are not dissimilar to the (1,1) front. Intuitively, this is because, under the previously described invariance due to scaling, these combinations are the same as (1,2) and (2,1) respectively. Such a difference may not be drastic enough to yield large changes to the Pareto fronts.

We recall from the main paper that for instance (6,20) the static solutions can be split into two groups: those without a copy of component 1, and those with a copy. We remarked previously that the static solutions with a copy of component 1 all became dominated by some dynamic policy. Figure A.3.13 shows that this remains true across all cases for the multipliers. However, in some cases we see other DOP solutions become dominated as well. Case (1,10) sees two more DOP solutions become dominated by dynamic policies. Case (5,1) sees one extra DOP solution become dominated, which is especially interesting as this solution must use a design which includes component 1. So in this case, not only do we see DOP solutions with component 1 become dominated by dynamic policies applied to designs without that component, but we also see the opposite; that is, previously non-dominated DOP solutions without a copy of component 1 become dominated by a dynamic policy applied to a design that includes component 1. Case (10,1) takes this to the extreme, and sees all except three DOP solutions become dominated by dynamic policies. This suggests that the benefits of using dynamic policies may become stronger in situations where component types differ greatly with respect to frequencies of events occurring.

# Appendix B

# Appendix for Chapter 5

## B.1 Preliminaries

In this section we provide a quick summary of the topic of equivalence relations. Much of the analysis in Chapter 5 relies on the idea of states or state-action pairs of a CTMDP being effectively (but not exactly) the same, and as such we rely on equivalence relations. We introduce our notation here. If $\sim$ is an equivalence relation, we write $s \sim t$ if $s$ is equivalent to $t$ under $\sim$. The *equivalence class* of $s$ can be denoted by $\widehat{s}$. If $\sim$ is an equivalence class over a set $\mathcal{S}$, we denote the set of equivalence classes by $\widehat{\mathcal{S}}$. We call the function $\varphi : \mathcal{S} \to \widehat{\mathcal{S}}$ that maps every element $s$ of the base set $\mathcal{S}$ to its equivalence class $\widehat{s}$ in $\widehat{\mathcal{S}}$ the *canonical surjection*. To compare the "strength" of equivalence relations, for two equivalence relations $\sim$ and $\sim'$ we say that $\sim$ *as strong as* $\sim'$ if and only if $s \sim t \implies s \sim' t$ for all $s, t \in \mathcal{S}$. Some literature would refer to this as $\sim$ being "finer" than $\sim'$.

## B.2 Proofs for Section 5.2

### B.2.1 Proof of Proposition 5.2.1

To show that $\{\widehat{S_k^\mu}\}_{k=1}^\infty$ is a Markov chain, it is enough to show that:

$$\mathbb{P}(\widehat{S_{k+1}^\mu} = \widehat{s}' | S_k^\mu = s) = \mathbb{P}(\widehat{S_{k+1}^\mu} = \widehat{s}' | \widehat{S_k^\mu} = \widehat{s}). \tag{B.2.1}$$

We first have the following:

$$\begin{aligned}\mathbb{P}(\widehat{S_{k+1}^\mu} = \widehat{s}' | S_k^\mu = s) &= \mathbb{P}(S_{k+1}^\mu \in \widehat{s}' | S_k^\mu = s) \\ &= \sum_{s'' \in \widehat{s}'} \mathbb{P}(S_{k+1}^\mu = s'' | S_k^\mu = s).\end{aligned}$$

There are two cases: $s \in \widehat{s}'$ (i.e. $\widehat{s} = \widehat{s}'$) and $s \notin \widehat{s}'$. In the first case we have the probability of nothing happening plus the probabilities of transitioning from $s$ to a different state in $\widehat{s}'$:

$$\begin{aligned}\mathbb{P}(\widehat{S_{k+1}^\mu} = \widehat{s} | S_k^\mu = s) &= (1 + q(s,s)\Delta) + \sum_{s'' \in \widehat{s}: s'' \neq s} q(s,s'')\Delta \\ &= 1 + (q^+(s,s) + q^-(s,s))\Delta + \sum_{s'' \in \widehat{s}: s'' \neq s} (q^+(s,s'') + q^-(s,s''))\Delta \\ &= 1 + \sum_{s'' \in \widehat{s}} (q^+(s,s'') + q^-(s,s''))\Delta \\ &= 1 + q^+(\widehat{s},\widehat{s})\Delta + q^-(s,s)\Delta \\ &= 1 + q^+(\widehat{s},\widehat{s})\Delta + q^-(\widehat{s},\widehat{s})\Delta \\ &= 1 + q(\widehat{s},\widehat{s})\Delta.\end{aligned}$$

Hence, $\mathbb{P}(\widehat{S^\mu_{k+1}} = \widehat{s} | S^\mu_k = s)$ depends only on $\widehat{s}$, so we can write:

$$\mathbb{P}(\widehat{S^\mu_{k+1}} = \widehat{s} | S^\mu_k = s) = \mathbb{P}(\widehat{S^\mu_{k+1}} = \widehat{s}' | \widehat{S^\mu_k} = \widehat{s}).$$

In the second case, we have:

$$\begin{aligned}\mathbb{P}(\widehat{S^\mu_{k+1}} = \widehat{s}' | S^\mu_k = s) &= \sum_{s'' \in \widehat{s}'} q(s, s'')\Delta \\ &= \sum_{s'' \in \widehat{s}'} q(\widehat{s}, s'')\Delta \\ &= q(\widehat{s}, \widehat{s}')\Delta,\end{aligned}$$

so (B.2.1) holds for both cases, with the stated transition probabilities.

### B.2.2 Proof of Corollary 5.2.1

We use the infinitesimal definition of the CTMC to prove that $\widehat{S^\mu(t)}$ is a CTMC. A continuous-time stochastic process $X(t)$ is a continuous-time Markov chain with initial distribution $\lambda$ and transition rate matrix $Q$ if $\mathbb{P}(X(0) = s) = \lambda(s)$ for all $s$, and for all states $s$,$s'$, small $h > 0$, and $t$ such that $\mathbb{P}(X(t) = s) > 0$, we have $\mathbb{P}(X(t+h) = s' | X(t) = s) = \delta_{s,s'} + q(s, s')h + o(h)$, where $\delta_{s,s'} = \mathbb{I}\{s = s'\}$ is the Kronecker delta, and $o(h)$ denotes a function $f(h)$ that grows slowly as $h \to 0$, i.e. $f(h)/h \to 0$ as $h \to 0$.

Let $S^\mu(t)$ be our controlled CTMDP, with initial distribution $\lambda$. Then we see:

$$\begin{aligned}\mathbb{P}(\widehat{S^\mu}(0) = \widehat{s}) &= \mathbb{P}(S^\mu(0) \in \widehat{s}) \\ &= \sum_{s' \in \widehat{s}} \mathbb{P}(S^\mu(0) = s') \\ &= \sum_{s' \in \widehat{s}} \lambda(s').\end{aligned}$$

So $\widehat{S^\mu}(t)$ has initial distribution $\widehat{\lambda}(\widehat{s}) = \sum_{s' \in \widehat{s}} \lambda(s)$. We now consider the expression

$\mathbb{P}(\widehat{S^\mu}(t+h) = \widehat{s}' | \widehat{S^\mu}(t) = \widehat{s})$. We first consider the case $\widehat{s} \neq \widehat{s}'$, and see that:

$$\begin{aligned}\mathbb{P}(\widehat{S^\mu}(t+h) = \widehat{s}' | \widehat{S^\mu}(t) = \widehat{s}) &= \mathbb{P}(S^\mu(t+h) \in \widehat{s}' | S^\mu(t+h) \in \widehat{s}) \\ &= \sum_{s'' \in \widehat{s}'} \mathbb{P}(S^\mu(t+h) = s'' | S^\mu(t+h) \in \widehat{s}).\end{aligned}$$

Now, as the equivalence classes $\widehat{s}$ are defined by an equivalence relation at least as strong as $\sim^{q^+}$, and $s'' \neq \widehat{s}$, we have:

$$\begin{aligned}\sum_{s'' \in \widehat{s}'} \mathbb{P}(S^\mu(t+h) = s'' | S^\mu(t+h) \in \widehat{s}) &= \sum_{s'' \in \widehat{s}'} \mathbb{P}(S^\mu(t+h) = s'' | S^\mu(t+h) = s), \\ &\qquad \text{for any } s \in \widehat{s} \\ &= \sum_{s'' \in \widehat{s}'} \left(q(s, s'')h + o(h)\right), \text{ for any } s \in \widehat{s} \\ &= \sum_{s'' \in \widehat{s}'} \left(q(\widehat{s}, s'')h + o(h)\right) \\ &= q(\widehat{s}, \widehat{s}')h + o(h).\end{aligned}$$

As such, we must have:

$$\mathbb{P}(\widehat{S^\mu}(t+h) = \widehat{s} | \widehat{S^\mu}(t) = \widehat{s}) = 1 + q(\widehat{s}, \widehat{s})h + o(h).$$

Therefore $\widehat{S^\mu(t)}$ is a CTMC with initial distribution $\widehat{\lambda}$ and transition rates $q(\widehat{s}, \widehat{s}')$.

### B.2.3 Proof of Proposition 5.2.2

We know that a stationary distribution $\boldsymbol{\pi}^\mu$ of $\mathfrak{C}$ controlled by policy $\mu$ satisfies the following:

$$\sum_{s \in \mathcal{S}} q_\mu(s, s')\pi^\mu(s) = 0,$$

for all $s' \in \mathcal{S}$. By splitting up the states by class and using $q = q^+ + q^-$, we see that:

$$\begin{aligned}\sum_{s\in\mathcal{S}} q_\mu(s,s')\pi^\mu(s) &= \sum_{\widehat{s}\in\widehat{S}}\sum_{s\in\widehat{s}} q^+_\mu(s,s')\pi^\mu(s) + \sum_{\widehat{s}\in\widehat{S}}\sum_{s\in\widehat{s}} q^-_\mu(s,s')\pi^\mu(s)\\ &= \sum_{\widehat{s}\in\widehat{S}}\sum_{s\in\widehat{s}} q^+_\mu(s,s')\pi^\mu(s) + q^-_\mu(s',s')\pi^\mu(s'),\end{aligned}$$

where the second equality holds because $q^-_\mu(\cdot, s')$ is zero everywhere except $s'$. Now recognise that by equivalence, $q^+_\mu(s,\cdot)$ depends only on the class $\widehat{s}$ of $s$, so we can write:

$$\sum_{\widehat{s}\in\widehat{S}}\sum_{s\in\widehat{s}} q^+_\mu(s,s')\pi^\mu(s) + q^-_\mu(s',s')\pi^\mu(s') = \sum_{\widehat{s}\in\widehat{S}} q^+_\mu(\widehat{s},s')\widehat{\pi^\mu}(\widehat{s}) + q^-_\mu(s',s')\pi^\mu(s'),$$

where $\widehat{\pi^\mu}(\widehat{s})$ follows the definition in the Proposition. As this holds for any target state $s'$, we can sum over all states $s'' \in \widehat{s'}$ in the equivalence class of $s'$, to obtain:

$$\begin{aligned}\sum_{s''\in\widehat{s'}}\sum_{\widehat{s}\in\widehat{S}} q^+_\mu(\widehat{s},s')\widehat{\pi^\mu}(\widehat{s}) + \sum_{s''\in\widehat{s'}} q^-_\mu(s'',s'')\pi^\mu(s'') &= \sum_{\widehat{s}\in\widehat{S}} q^+_\mu(\widehat{s},\widehat{s'})\widehat{\pi^\mu}(\widehat{s}) + q^-_\mu(\widehat{s'},\widehat{s'})\widehat{\pi^\mu}(\widehat{s'})\\ &= \sum_{\widehat{s}\in\widehat{S}} q^+_\mu(\widehat{s},\widehat{s'})\widehat{\pi^\mu}(\widehat{s}) + \sum_{\widehat{s}\in\widehat{S}} q^-_\mu(\widehat{s},\widehat{s'})\widehat{\pi^\mu}(\widehat{s'})\\ &= \sum_{\widehat{s}\in\widehat{S}} q_\mu(\widehat{s},\widehat{s'})\widehat{\pi^\mu}(\widehat{s})\\ &= 0.\end{aligned}$$

Therefore we have $\sum_{\widehat{s}\in\widehat{S}} q_\mu(\widehat{s},\widehat{s'})\widehat{\pi^\mu}(\widehat{s}) = 0$ for all $\widehat{s'} \in \widehat{\mathcal{S}}$, so $\widehat{\pi^\mu}(\widehat{s})$ gives the stationary distribution of the class-aggregated CTMC.

### B.2.4 Proof of Lemma 5.2.1

For any state $s$, we see that:

$$\begin{aligned}
\widehat{\pi}(\widehat{s, \mu(s)}) &= \sum_{(s',a') \in (\widehat{s,\mu(s)})} \pi(s', a') \\
&= \sum_{s' \in \mathcal{S}: (s', \mu(s') \in (\widehat{s,\mu(s)})} \pi(s', \mu(s')) \text{ (at most one non-zero term per state)} \\
&= \sum_{s' \in \mathcal{S}: (s', \mu(s') \in (\widehat{s,\mu(s)})} \pi^{\mu}(s') \\
&= \widehat{\pi^{\mu}}(\widehat{s}).
\end{aligned}$$

Now it's clear that the $\widehat{\pi}(\widehat{s, \mu(s)})$ alone sum to one, so $\widehat{\pi}(\widehat{s,a}) = 0$ for all state action equivalence classes $\widehat{s,a}$ that can't be written in the form $\widehat{s', \mu(s')}$ for some $s' \in \mathcal{S}$.

### B.2.5 Effect of Sequencing for Sub-Optimal Policies

**Proposition B.2.1.** *Let $\mu^*$ be an average-optimal policy for an impulsive CTMDP. Let $J_t^{\mu^*}$ be the finite-horizon value function under policy $\mu^*$ for state $s$ until time $t$. Assume $t > T$ for some $T$ large enough that $\mu^*$ is $t$-optimal for all $t > T$, and $T \geq t(s,a)$ for all $s, a$. Let $Q_t^{\mu^*}$ be the action value function. Let $a_1 \neq 0$ be an action for state $s$ satisfying $Q_t^{\mu^*}(s, a_1) = (1 + \varepsilon(t)) J_t^{\mu^*}(s)$ for some $\varepsilon(t) \geq 0$, and let $a_2 \neq 0$ be an action for $s \oplus a_1$ satisfying $Q_t^{\mu^*}(s \oplus a_1, a_2) = (1 + \varepsilon'(t)) J_t^{\mu^*}(s \oplus a_1)$ for some $\varepsilon'(t) \geq 0$. Then $Q_t^{\mu^*}(s, a_1 \oplus a_2) \leq (1 + \varepsilon(t))(1 + \varepsilon'(t)) J_t^{\mu^*}(s)$. Additionally, there exist cases where this bound is tight.*

*Proof.* We see:

$$\begin{aligned}
Q_t^{\mu^*}(s, a_1 \oplus a_2) &= Q_t^{\mu^*}(s \oplus a_1, a_2) && \text{(by impulsivity)} \\
&= (1 + \varepsilon'(t)) J_t^{\mu^*}(s \oplus a_1) && \text{(by } \varepsilon'\text{-bound)} \\
&\leq (1 + \varepsilon'(t)) Q_t^{\mu^*}(s \oplus a_1, 0) && \text{(by optimality of } J_t^{\mu^*}\text{)} \\
&= (1 + \varepsilon'(t)) Q_t^{\mu^*}(s, a_1) && \text{(by impulsivity)} \\
&= (1 + \varepsilon(t))(1 + \varepsilon'(t)) J_t^{\mu^*}(s) && \text{(by } \varepsilon\text{-bound).}
\end{aligned}$$

We can also demonstrate that this bound is tight. Consider the following impulsive CTMDP:

$\mathcal{S} = \{1, 2, 3\}$,

$\mathcal{A}(1) = \{0, 1, 2\}, \mathcal{A}(2) = \{0, 1\}, \mathcal{A}(3) = \{0\}$,

$q^+(1, 0, \cdot) = 0$, $q^+(2, 0, s') = \begin{cases} 1, & s' = 1, \\ 0, & \text{otherwise,} \end{cases}$ $q^+(3, 0, s') = \begin{cases} 1, & s' = 1, \\ 0, & \text{otherwise,} \end{cases}$

$c(1) = 1, c(2) > c(1), c(3) > c(2)$,

$s \oplus a = s + a$.

In words, state 1 is absorbing under the zero action and has the lowest cost rate, states 2 and 3 deterministically transition to state 1 after an expected sojourn time of 1, state 2 is more expensive than state 1, and state 3 is more expensive than state 2. Clearly, the average-optimal policy is $\mu^*(s) = 0$ for all $s$. In fact, this policy is optimal for all finite horizons, so we can write $V^*$ instead of $V^{\mu^*}$ and $Q^*$ instead of $Q^{\mu^*}$. We

may evaluate the finite-horizon value functions:

$$V^*(1;t) = Q^*(1,0;t) = t$$

$$V^*(2;t) = Q^*(2,0;t) = \begin{cases} c(2)t, & t \in [0,1], \\ c(2) + (t-1), & t > 1. \end{cases}$$

$$V^*(3;t) = Q^*(3,0;t) = \begin{cases} c(3)t, & t \in [0,1], \\ c(3) + (t-1), & t > 1. \end{cases}$$

In state 1, we always accumulate cost at rate 1. In state 2, we accumulate cost at rate $c(2)$ until time $t = 1$ and then transition to state 1 and accumulate cost at rate 1. State 3 is the same except that $c(2)$ is replaced by $c(3)$. We may also evaluate the remaining action value functions and rewrite them in terms of error functions $\varepsilon(s,a;t)$ where $Q^*(s,a;t) = (1+\varepsilon(s,a;t))V^*(s;t)$, with $\varepsilon(\cdot,\cdot;0) = 0$:

$$Q^*(1,1;t) = V^*(2;t) \text{ (by impulsivity)}$$

$$= (1+\varepsilon(1,1;t))V^*(1;t), \text{ for } \varepsilon(1,1;t) = \begin{cases} c(2)-1, & t \in (0,1], \\ \frac{c(2)-1}{t}, & t > 1, \end{cases}$$

$$Q^*(1,2;t) = V^*(3;t) \text{ (by impulsivity)}$$

$$= (1+\varepsilon(1,2;t))V^*(1;t), \text{ for } \varepsilon(1,2;t) = \begin{cases} c(3)-1, & t \in (0,1], \\ \frac{c(3)-1}{t}, & t > 1, \end{cases}$$

$$Q^*(2,1;t) = V^*(3;t) \text{ (by impulsivity)}$$

$$= (1+\varepsilon(2,1;t))V^*(2;t), \text{ for } \varepsilon(2,1;t) = \begin{cases} \frac{c(3)}{c(2)}-1, & t \in (0,1], \\ \frac{c(3)-c(2)}{c(2)-1+t}, & t > 1. \end{cases}$$

Some rearranging then shows that $(1+\varepsilon(1,2;t)) = (1+\varepsilon(1,1;t))(1+\varepsilon(2,1;t))$, where action 2 in state 1 is the sequenced action of action 1 in state 1 and action 1 in state

2. □

Proposition B.2.1 provides the rigorous statement and proof of Remark 5.2.3. Note that due to the reliance on $Q^{\mu^*}$, this provides a best-case scenario for the extent of the suboptimality, as it assumes that an optimal policy is followed thereafter. This also extends partially to the multi-objective case via mixed-objective scalarisation. If no DS policy with $\mu(s) = a_1$ and $\mu(s \oplus a_1) = a_2$ is Pareto-optimal, then for every weighting $\boldsymbol{w}$ there exists an optimal policy $\mu^*_{\boldsymbol{w}}$ such that $a_1$ and $a_2$ are sub-optimal in the way described, and therefore $a_1 \oplus a_2$ has worse optimality bounds in all of these cases.

### B.2.6 Commutativity of Projections and Canonical Surjections

The following proposition describes the idea that it makes sense to speak of the "$i^{\text{th}}$ compartment" of an impulsive equivalence class $\widehat{(\boldsymbol{s}, \boldsymbol{a})}$, and that this is identical to considering the impulsive equivalence class of the $i^{\text{th}}$ compartment of any $(\boldsymbol{s}, \boldsymbol{a}) \in \widehat{(\boldsymbol{s}, \boldsymbol{a})}$. Mathematically, we can say that assignment to equivalence classes and taking $i^{\text{th}}$ compartments commute. First, let us recall all necessary notation, and define some additional notation. Consider a stochastically decomposable impulsive CTMDP with feasible state-action space $\mathcal{SA}$, state-action impulsive equivalence relation $\sim_{\oplus}$ and compartment state-action impulsive equivalence relations $\sim_{\oplus_i}$. Let $\widehat{\mathcal{SA}}$ be the partition of $\mathcal{SA}$ induced by $\sim_{\oplus}$, $\mathcal{SA}_i$ be the $i^{\text{th}}$ compartment state-action space for each $i$, and $\widehat{\mathcal{SA}_i}$ be the partitions of the $\mathcal{SA}_i$ induced by the respective relations $\sim_{\oplus_i}$. Let $\varphi : \mathcal{SA} \to \widehat{\mathcal{SA}}$ be the canonical surjection of $\sim_{\oplus}$, i.e. the function that maps each state-action pair $(\boldsymbol{s}, \boldsymbol{a})$ to its equivalence class, so $\varphi(\boldsymbol{s}, \boldsymbol{a}) = \widehat{(\boldsymbol{s}, \boldsymbol{a})}$. Likewise, let $\varphi_i : \mathcal{SA}_i \to \widehat{\mathcal{SA}_i}$ be the canonical surjections of the $\sim_{\oplus_i}$. Let $\chi_i : \mathcal{SA} \to \mathcal{SA}_i$ be the $i^{\text{th}}$ projection map of $\mathcal{SA}$ onto $\mathcal{SA}_i$, i.e. $\chi_i(\boldsymbol{s}, \boldsymbol{a}) = ([\boldsymbol{s}]_i, [\boldsymbol{a}]_i)$, the function that maps each state-action pair to its $i^{\text{th}}$ compartment.

**Proposition B.2.2.** *For each $i$, there exists a projection $\psi_i : \widehat{\mathcal{SA}} \to \widehat{\mathcal{SA}_i}$ such that for every $\widehat{(\boldsymbol{s}, \boldsymbol{a})} \in \widehat{\mathcal{SA}}$ and $(\boldsymbol{s}, \boldsymbol{a}) \in \widehat{(\boldsymbol{s}, \boldsymbol{a})}$, $\varphi_i([\boldsymbol{s}]_i, [\boldsymbol{a}]_i) = \psi_i(\widehat{(\boldsymbol{s}, \boldsymbol{a})})$.*

*Proof.* Let $\widehat{(\boldsymbol{s}, \boldsymbol{a})} \in \widehat{\mathcal{SA}}$, and choose any $(\boldsymbol{s}_1, \boldsymbol{a}_1), (\boldsymbol{s}_2, \boldsymbol{a}_2) \in \widehat{(\boldsymbol{s}, \boldsymbol{a})}$. By Lemma 5.2.7, we have $[\boldsymbol{s}_1, \boldsymbol{a}_1]_i \sim_{\oplus_i} [\boldsymbol{s}_2, \boldsymbol{a}_2]_i$ for every $i$, so $\varphi_i([\boldsymbol{s}_1, \boldsymbol{a}_1]_i) = \varphi_i([\boldsymbol{s}_2, \boldsymbol{a}_2]_i)$ for every $i$. Therefore any $\widehat{(\boldsymbol{s}, \boldsymbol{a})} \in \widehat{\mathcal{SA}}$ can be mapped to exactly one member of $\widehat{\mathcal{SA}}_i$ for every $i$, and we can denote this by $\psi_i(\widehat{(\boldsymbol{s}, \boldsymbol{a})})$. □

Diagrammatically, we say that the following diagram commutes for all $i$:

$$\begin{array}{ccc} \mathcal{SA} & \xrightarrow{\varphi} & \widehat{\mathcal{SA}} \\ {\scriptstyle \chi_i}\downarrow & & \downarrow{\scriptstyle \psi_i} \\ \mathcal{SA}_i & \xrightarrow{\varphi_i} & \widehat{\mathcal{SA}}_i \end{array}$$

The following corollary is equivalent to the above, but for a fixed separable policy $\mu$. Again, we first recall all necessary notation. Consider the whole-state $\mu$-adapted impulsive equivalence relation $\sim_{\oplus\mu}$ and compartment state $\mu$-adapted impulsive equivalence relations $\sim_{\oplus\mu_i}$. Let $\widehat{\mathcal{S}}$ be the partition of $\mathcal{S}$ induced by $\sim_{\oplus\mu}$, and $\widehat{\mathcal{S}}_i$ be the partitions of the $\mathcal{S}_i$ induced by the relations $\sim_{\oplus\mu_i}$. Let $\varphi^\mu : \mathcal{S} \to \widehat{\mathcal{S}}$ be the canonical surjection of $\sim_{\oplus\mu}$, $\varphi(\boldsymbol{s}) = \widehat{\boldsymbol{s}}$. Likewise, let $\varphi_i^\mu : \mathcal{S}_i \to \widehat{\mathcal{S}}_i$ be the canonical surjections of the $\sim_{\oplus\mu_i}$. Let $\chi_i^\mu : \mathcal{S} \to \mathcal{S}_i$ be the $i$th projection map of $\mathcal{S}$ onto $\mathcal{S}_i$, i.e. $\chi_i^\mu(\boldsymbol{s}) = [\boldsymbol{s}]_i$.

**Corollary B.2.1.** *For each $i$, there exists a projection $\psi_i^\mu : \widehat{\mathcal{S}} \to \widehat{\mathcal{S}}_i$ such that for every $\widehat{\boldsymbol{s}} \in \widehat{\mathcal{S}}$ and $\boldsymbol{s} \in \widehat{\boldsymbol{s}}$, $\varphi_i^\mu([\boldsymbol{s}]_i) = \psi_i^\mu(\widehat{\boldsymbol{s}})$.*

*Proof.* Let $\widehat{s} \in \widehat{\mathcal{S}}$, and choose any $\boldsymbol{s}_1, \boldsymbol{s}_2 \in \widehat{\boldsymbol{s}}$. By Corollary 5.2.4, we have $[\boldsymbol{s}_1]_i \sim_{\oplus\mu_i} [\boldsymbol{s}_2]_i$ for every $i$, so $\varphi_i^\mu([\boldsymbol{s}_1]_i) = \varphi_i^\mu([\boldsymbol{s}_2]_i)$ for every $i$. Therefore any $\widehat{\boldsymbol{s}} \in \widehat{\mathcal{S}}$ can be mapped to exactly one member of $\widehat{\mathcal{S}}_i$ for every $i$, and we can denote this by $\psi_i^\mu(\widehat{\boldsymbol{s}})$. □

Diagrammatically, we say that the following diagram commutes for all $i$:

$$\begin{array}{ccc} \mathcal{S} & \xrightarrow{\varphi^\mu} & \widehat{\mathcal{S}} \\ {\scriptstyle \chi_i^\mu}\downarrow & & \downarrow{\scriptstyle \psi_i^\mu} \\ \mathcal{S}_i & \xrightarrow{\varphi_i^\mu} & \widehat{\mathcal{S}}_i \end{array}$$

### B.2.7 Proof of Proposition 5.2.6

For any $i$, we want to show that:

$$\mathbb{P}(\widehat{S^{\mu}_{i,k+1}} = \widehat{v} | S^{\mu}_{k} = \boldsymbol{s}) = \mathbb{P}(\widehat{S^{\mu}_{i,k+1}} = \widehat{v} | \widehat{S^{\mu}_{i,k}} = \widehat{[s]}_i),$$

where $v \in \mathcal{S}_i$. To begin, we see that:

$$\begin{aligned}\mathbb{P}(S^{\mu}_{i,k+1} = v | S^{\mu}_{k} = \boldsymbol{s}) &= \mathbb{P}(S^{\mu}_{k+1} \in \{\boldsymbol{s}' \in \mathcal{S} : [\boldsymbol{s}']_i = v\} | S^{\mu}_{k} = \boldsymbol{s}) \\ &= \sum_{\boldsymbol{s}' \in \mathcal{S} : [\boldsymbol{s}']_i = v} \mathbb{P}(S^{\mu}_{k+1} = \boldsymbol{s}' | S^{\mu}_{k} = \boldsymbol{s}).\end{aligned}$$

There are three main cases to consider: (i) the probability of nothing happening, in which case $v = [\boldsymbol{s}]_i$; (ii) the probability of something happening in another sub-state, in which case nothing happens to the $i$th sub-state except being updated to reflect the impulsive action, so $v = [\boldsymbol{s} \oplus \mu(\boldsymbol{s})]_i$, and (iii) the probability that an event happens in compartment $i$. We need only focus on the latter case. We recall that for any state $\boldsymbol{s}$, if there is an event in compartment $i$ then all other compartments $j \neq i$ deterministically update, so that $[\boldsymbol{s}']_j = [\boldsymbol{s} \oplus \mu(\boldsymbol{s})]_j$. Hence, for any $v \notin \{[\boldsymbol{s}]_i, [\boldsymbol{s} \oplus \mu(\boldsymbol{s})]_i\}$ we can write:

$$\begin{aligned}\mathbb{P}(S^{\mu}_{i,k+1} = v | S^{\mu}_{k} = \boldsymbol{s}) &= \sum_{\boldsymbol{s}' \in \mathcal{S} : [\boldsymbol{s}']_i = v} \mathbb{P}(S^{\mu}_{k+1} = \boldsymbol{s}' | S^{\mu}_{k} = \boldsymbol{s}) \\ &= \mathbb{P}(S^{\mu}_{k+1} = \boldsymbol{s}'' | S^{\mu}_{k} = \boldsymbol{s}) \\ &= q(\boldsymbol{s}, \mu(\boldsymbol{s}), \boldsymbol{s}'')\Delta \\ &= q_{\mu_i}([\boldsymbol{s}]_i, v)\Delta, \text{ (assuming a separable policy)},\end{aligned}$$

where $\boldsymbol{s}'' = ([\boldsymbol{s}]_1 \oplus_1 \mu_1([\boldsymbol{s}]_1), \ldots, v, \ldots, [\boldsymbol{s}]_I \oplus_I \mu_I([\boldsymbol{s}]_I))$.

We now wish to consider $\widehat{S^{\mu}_{i,k+1}}$. Choose some equivalence class $\widehat{v} \neq \widehat{[\boldsymbol{s}]}_i$, and recognise that by coherence we have that $[\boldsymbol{s}]_i \sim_{\oplus\mu_i} [\boldsymbol{s} \oplus \mu(\boldsymbol{s})]_i$, so $[\boldsymbol{s}]_i \notin \widehat{v}$ and $[\boldsymbol{s} \oplus \mu(\boldsymbol{s})]_i \notin \widehat{v}$.

Then we can see:

$$\begin{aligned}
\mathbb{P}(\widehat{S^{\mu}_{i,k+1}} = \widehat{v} | S^{\mu}_{k} = \boldsymbol{s}) &= \mathbb{P}(S^{\mu}_{i,k+1} \in \widehat{v} | S^{\mu}_{k} = \boldsymbol{s}) \\
&= \sum_{v' \in \widehat{v}} \mathbb{P}(S^{\mu}_{i,k+1} = v' | S^{\mu}_{k} = s) \\
&= \sum_{v' \in \widehat{v}} q_{\mu_i}([\boldsymbol{s}]_i, v')\Delta \\
&= \sum_{v' \in \widehat{v}} q_{\mu_i}(\widehat{[\boldsymbol{s}]_i}, v')\Delta \\
&= q_{\mu_i}(\widehat{[s]_i}, \widehat{v})\Delta.
\end{aligned}$$

We note that this expression depends only on the sub-state equivalence classes, so we can therefore write:

$$\mathbb{P}(\widehat{S^{\mu}_{i,k+1}} = \widehat{v}' | \widehat{S^{\mu}_{i,k}} = \widehat{v}) = q_{\mu_i}(\widehat{v}, \widehat{v}')\Delta,$$

for any $\widehat{v}' \neq \widehat{v}$. We can then see that:

$$\begin{aligned}
\mathbb{P}(\widehat{S^{\mu}_{i,k+1}} = \widehat{v} | \widehat{S^{\mu}_{i,k}} = \widehat{v}) &= 1 - \mathbb{P}(\widehat{S^{\mu}_{i,k+1}} \neq \widehat{v} | \widehat{S^{\mu}_{i,k}} = \widehat{v}) \\
&= 1 - \sum_{\widehat{v}' \neq \widehat{v}} \mathbb{P}(\widehat{S^{\mu}_{i,k+1}} = \widehat{v}' | \widehat{S^{\mu}_{i,k}} = \widehat{v}) \\
&= 1 - \sum_{\widehat{v}' \neq \widehat{v}} q_{\mu_i}(\widehat{v}, \widehat{v}')\Delta \\
&= 1 + q_{\mu_i}(\widehat{v}, \widehat{v})\Delta.
\end{aligned}$$

Therefore $\mathbb{P}(\widehat{S^{\mu}_{i,k+1}} = \widehat{v} | S^{\mu}_{k} = \boldsymbol{s})$ depends only on the respective compartment equivalence class of $\boldsymbol{s}$, so the stochastic processes $\left\{\widehat{S^{\mu}_{i,k}}\right\}_{k=1}^{\infty}$ for each $i$ are independent Markov chains.

### B.2.8 Proof of Proposition 5.2.5

Let $\boldsymbol{g} : [0,1]^{|\mathcal{SA}|} \to \mathbb{R}^K, \boldsymbol{g}(\boldsymbol{\pi}) = \sum_{\boldsymbol{s}\in\mathcal{S}} \boldsymbol{c}(\boldsymbol{s} \oplus \boldsymbol{a})\pi(\boldsymbol{s},\boldsymbol{a})$ be the function mapping every solution of OP to its long-run average cost. First, assume $c_k$ is an additive cost function. We see:

$$
\begin{aligned}
g_k(\boldsymbol{\pi}) &= \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}} c_k(\boldsymbol{s}\oplus\boldsymbol{a})\pi(\boldsymbol{s},\boldsymbol{a}) \\
&= \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}} \left(\sum_{i=1}^{I} c_{i,k}([\boldsymbol{s}\oplus\boldsymbol{a}]_i)\right)\pi(\boldsymbol{s},\boldsymbol{a}) \\
&= \sum_{i=1}^{I} \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}} c_{i,k}([\boldsymbol{s}\oplus\boldsymbol{a}]_i)\pi(\boldsymbol{s},\boldsymbol{a}) \\
&= \sum_{i=1}^{I} \sum_{(s_i,a_i)\in\mathcal{SA}_i} \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}:[\boldsymbol{s},\boldsymbol{a}]_i=(s_i,a_i)} c_{i,k}(s_i\oplus a_i)\pi(\boldsymbol{s},\boldsymbol{a}) \\
&= \sum_{i=1}^{I} \sum_{(s_i,a_i)\in\mathcal{SA}_i} c_{i,k}(s_i\oplus a_i)\pi_i(s_i,a_i).
\end{aligned}
$$

Therefore $c_k$ is decomposable with $f_k : \mathbb{R}^I \to \mathbb{R}$, $f_k(\boldsymbol{x}) = \sum_{i=1}^{I} x_i$. Next, assume $c_k$ is a multiplicative cost function. We see:

$$\begin{aligned}
g_k(\pi) &= \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}} c_k(\boldsymbol{s}\oplus\boldsymbol{a})\pi(\boldsymbol{s},\boldsymbol{a}) \\
&= \sum_{\widehat{\boldsymbol{s},\boldsymbol{a}}\in\widehat{\mathcal{SA}}} c_k(\widehat{\boldsymbol{s},\boldsymbol{a}})\widehat{\pi}(\widehat{\boldsymbol{s},\boldsymbol{a}}) \\
&= \sum_{\widehat{\boldsymbol{s},\boldsymbol{a}}\in\widehat{\mathcal{SA}}} \left(\prod_{i=1}^{I} c_{i,k}([\widehat{\boldsymbol{s},\boldsymbol{a}}]_i)\right)\left(\prod_{i=1}^{I} \widehat{\pi}_i([\widehat{\boldsymbol{s},\boldsymbol{a}}]_i)\right) \\
&\qquad \text{(by independence of sub-processes under separable } \mu) \\
&= \sum_{\widehat{\boldsymbol{s},\boldsymbol{a}}\in\widehat{\mathcal{SA}}} \prod_{i=1}^{I} c_{i,k}([\widehat{\boldsymbol{s},\boldsymbol{a}}]_i)\widehat{\pi}_i([\widehat{\boldsymbol{s},\boldsymbol{a}}]_i) \\
&= \prod_{i=1}^{I} \sum_{\widehat{s_i,a_i}\in\widehat{SA}_i} c_{i,k}(\widehat{s_i,a_i})\pi_i(\widehat{s_i,a_i}) \\
&= \prod_{i=1}^{I} \sum_{s_i,a_i\in\mathcal{SA}_i} c_{i,k}(s_i,a_i)\pi_i(s_i,a_i).
\end{aligned}$$

Therefore $c_k$ is decomposable with $f_k : \mathbb{R}^I \to \mathbb{R}$, $f_k(\boldsymbol{x}) = \prod_{i=1}^{I} x_i$. Finally, assume $c_k$ is a logical-OR cost function. We first see that:

$$\begin{aligned}
c_k(\boldsymbol{s}) &= \mathbb{I}\left\{\bigvee_{i=1}^{I}([\boldsymbol{s}]_i \in E_i)\right\} \\
&= \mathbb{I}\left\{\neg\neg\bigvee_{i=1}^{I}([\boldsymbol{s}]_i \in E_i)\right\} \\
&= 1 - \mathbb{I}\left\{\bigwedge_{i=1}^{I}([\boldsymbol{s}]_i \notin E_i)\right\} \\
&= 1 - \prod_{i=1}^{I}\mathbb{I}\left\{[\boldsymbol{s}]_i \notin E_i\right\}.
\end{aligned}$$

We then see that:

$$\begin{aligned} g_k(\boldsymbol{\pi}) &= \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}} c_k(\boldsymbol{s}\oplus\boldsymbol{a})\pi(\boldsymbol{s},\boldsymbol{a}) \\ &= \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}} \left(1-\prod_{i=1}^{I}\mathbb{I}\left\{[\boldsymbol{s}]_i\notin E_i\right\}\right)\pi(\boldsymbol{s},\boldsymbol{a}) \\ &= 1-\sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}} \left(\prod_{i=1}^{I}\mathbb{I}\left\{[\boldsymbol{s}]_i\notin E_i\right\}\right)\pi(\boldsymbol{s},\boldsymbol{a}). \end{aligned}$$

We then treat the sum along similar lines to the multiplicative cost function to find that:

$$\begin{aligned} g_k(\pi) &= 1-\prod_{i=1}^{I}\sum_{s_i,a_i\in\mathcal{SA}_i}\mathbb{I}\left\{s_i\notin E_i\right\}\pi_i(s_i,a_i) \\ &= 1-\prod_{i=1}^{I}\left(1-\sum_{s_i,a_i\in\mathcal{SA}_i}\mathbb{I}\left\{s_i\in E_i\right\}\pi_i(s_i,a_i)\right), \end{aligned}$$

so $c_k$ is decomposable with $c_{i,k}(s_i) = \mathbb{I}\{s_i\in E_i\}$ and $f_k : [0,1]^I \to [0,1]$, $f(\boldsymbol{x}) = 1-\prod_{i=1}^{I}(1-x_i)$.

## B.3 Proof of Theorem 5.3.1

We first show that it is possible to move from CS-MILP solutions to DC-MIP solutions in such a way that equations (5.2.8) and (5.2.9) are satisfied, where equations (5.2.8) and (5.2.9) are the decision variable equivalent of Corollary 5.2.6.

**Proposition B.3.1.** *Let $\mathfrak{C}$ be a decomposable impulsive CTMDP, and choose a decomposition into compartments $\{\mathfrak{C}_i\}_{i=1}^I$. Let $(\boldsymbol{\pi},\boldsymbol{y})$ be a solution to CS-MILP for $\mathfrak{C}$. Then there exists a solution $\{\boldsymbol{\pi}_i,\boldsymbol{y}_i\}_{i=1}^I$ for DC-MIP with compartments $\{\mathfrak{C}_i\}_{i=1}^I$ satisfying the equations (5.2.8) and (5.2.9).*

*Proof.* $\pi(\boldsymbol{s},\boldsymbol{a})$ is a basic feasible solution of OP for $\mathfrak{C}$, and corresponds to a determin-

istic policy $\mu(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}\in\mathcal{A}(\boldsymbol{s})} \pi(\boldsymbol{s},\boldsymbol{a})$. Moreover, by the separable and coherent constraints, $\mu$ must be a separable and coherent policy. Therefore, $\mu$ can be written as $(\mu_1,\ldots,\mu_I)$ where the $\mu_i$ are coherent policies for the compartments $\mathfrak{C}_i$. Now, let $V^{\mu_i}$ be the stationary distribution of the CTMC defined by compartment $\mathfrak{C}_i$ as controlled by policy $\mu_i$, for each $i$. Then:

$$\pi_i(v,b) := \begin{cases} \mathbb{P}(V^{\mu_i} = v), & b = \mu_i(v) \\ 0, & \textit{otherwise}, \end{cases}.$$

provides a basic feasible solution for OP for $\mathcal{C}_i$, satisfies the equivalent constraints in DC-MIP, and satisfies (5.2.8) and (5.2.9). If we let $y_i(v,b) = \mathbb{I}\{b = \mu_i(v)\}$ for all $i$, and recall that the $\mu_i$ are coherent, then $\{\boldsymbol{\pi}_i, \boldsymbol{y}_i\}_{i=1}^I$ provides a feasible solution to DC-MIP. □

The next result shows that the same approach works in the other direction, moving from solutions of DC-MIP to solutions of CS-MILP.

**Proposition B.3.2.** *Let $\mathfrak{C}$ be a decomposable impulsive CTMDP, and choose a decomposition into compartments $\{\mathfrak{C}_i\}_{i=1}^I$. Let $\{\boldsymbol{\pi}_i, \boldsymbol{y}_i\}_{i=1}^I$ be a solution to DC-MIP for compartments $\{\mathfrak{C}_i\}_{i=1}^I$. Then there exists a solution $(\boldsymbol{\pi}, \boldsymbol{y})$ to CS-MILP for $\mathfrak{C}$ satisfying (5.2.8) and (5.2.9).*

*Proof.* The $\boldsymbol{\pi}_i$ correspond to basic feasible solutions of OP for each $\mathfrak{C}_i$, which in turn corresponds to coherent policies $\mu_i(v) = \arg\max_{b\in\mathcal{A}_i(v)} \pi_i(v,b)$. Together, these form the CS policy $\mu = (\mu_1,\ldots,\mu_I)$ for the whole CTMDP $\mathfrak{C}$. Let $S^\mu$ be the stationary distribution of $\mathfrak{C}$ controlled by $\mu$. Now we define:

$$\pi(\boldsymbol{s},\boldsymbol{a}) := \begin{cases} \mathbb{P}(S^\mu = \boldsymbol{s}), & \boldsymbol{a} = \mu(\boldsymbol{s}), \\ 0, & \text{otherwise}. \end{cases}$$

Then $\pi(\boldsymbol{s}, \boldsymbol{a})$ provides a basic feasible solution to OP. Together with $y(\boldsymbol{s}, \boldsymbol{a}) := \mathbb{I}\{\boldsymbol{a} = \mu(\boldsymbol{s})\}$, $(\boldsymbol{\pi}, \boldsymbol{y})$ gives a solution to CS-MILP that satisfies the relevant equations. □

As a result, we can move from solutions of CS-MILP to DC-MIP and vice versa in such a way that the relevant equations are satisfied. We now bring these together to prove Theorem 5.3.1.

*Proof.* We must show that the existence of a solution with a certain objective value in one of the problems implies the existence of a solution in the other problem with the same objective value.

First, let $(\boldsymbol{\pi}, \boldsymbol{y})$ be a solution to CS-MILP. We want to show that it has a corresponding solution in DC-MIP with the same objective value. The solution $(\boldsymbol{\pi}, \boldsymbol{y})$ can be mapped to a solution $(\boldsymbol{\pi}_i, \boldsymbol{y}_i)_{i=1}^I$ of DC-MIP by Proposition B.3.1, with $\boldsymbol{\pi}$ and $\boldsymbol{\pi}_i$ related by (5.2.8) and (5.2.9). We recall:

$$g_k(\boldsymbol{\pi}) = f_k\left(\left(\sum_{s_1, a_1 \in \mathcal{SA}_1} c_{1,k}(s_1 \oplus a_1)\pi_1(s_1, a_1)\right)_{i=1}^{I}\right),$$

and we need only focus on one entry in this, so for any compartment $i$ and objective $k$

we have:

$$\begin{aligned}
\sum_{(s_i,a_i)\in\mathcal{SA}_i} c_{i,k}(s_i \oplus a_i)\pi_i(s_i,a_i) &= \sum_{(s_i,a_i)\in\mathcal{SA}_i} \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}:[\boldsymbol{s},\boldsymbol{a}]_i=(s_i,a_i)} c_{i,k}(s_i \oplus a_i)\pi(\boldsymbol{s},\boldsymbol{a}) \\
&= \sum_{\widehat{s_i,a_i}\in\widehat{\mathcal{SA}_i}} \sum_{(s_i,a_i)\in\widehat{s_i,a_i}} \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}:[\boldsymbol{s},\boldsymbol{a}]_i=(s_i,a_i)} c_{i,k}(s_i \oplus a_i)\pi(\boldsymbol{s},\boldsymbol{a}) \\
&= \sum_{\widehat{s_i,a_i}\in\widehat{\mathcal{SA}_i}} c_{i,k}(\widehat{s_i,a_i}) \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{SA}:[\boldsymbol{s},\boldsymbol{a}]_i\in\widehat{s_i,a_i}} \pi(\boldsymbol{s},\boldsymbol{a}) \\
&= \sum_{\widehat{s_i,a_i}\in\widehat{\mathcal{SA}_i}} c_{i,k}(\widehat{s_i,a_i}) \sum_{\widehat{\boldsymbol{s},\boldsymbol{a}}\in\widehat{\mathcal{SA}}} \sum_{\boldsymbol{s},\boldsymbol{a}\in\widehat{s,a}:[\boldsymbol{s},\boldsymbol{a}]_i\in\widehat{s_i,a_i}} \pi(\boldsymbol{s},\boldsymbol{a}) \\
&= \sum_{\widehat{s_i,a_i}\in\widehat{\mathcal{SA}_i}} c_{i,k}(\widehat{s_i,a_i}) \sum_{\widehat{\boldsymbol{s},\boldsymbol{a}}\in\widehat{\mathcal{SA}}:[\widehat{\boldsymbol{s},\boldsymbol{a}}]_i=\widehat{s_i,a_i}} \widehat{\pi}(\widehat{\boldsymbol{s},\boldsymbol{a}}) \\
&= \sum_{\widehat{s_i,a_i}\in\widehat{\mathcal{SA}_i}} c_{i,k}(\widehat{s_i,a_i})\widehat{\pi}_i(\widehat{s_i,a_i}).
\end{aligned}$$

Therefore $\boldsymbol{g}(\boldsymbol{\pi})$ is entirely determined by the $\pi_i(\widehat{s_i,a_i})$ as determined by the equivalent solution to DC-MIP, so every solution to CS-MILP has a solution in DC-MIP with the same objective value.

Secondly, we want to show the other direction. Let $(\boldsymbol{\pi}_i,\boldsymbol{y}_i)_{i=1}^I$ be a solution to DC-MIP. We want to show that it has a corresponding solution in CS-MILP with the same objective value. The solution $(\boldsymbol{\pi}_i,\boldsymbol{y}_i)_{i=1}^I$ can be mapped to a solution $(\boldsymbol{\pi},\boldsymbol{y})$ of CS-MILP by Proposition B.3.2, with $\boldsymbol{\pi}$ and $\boldsymbol{\pi}_i$ related by (5.2.8) and (5.2.9). Starting with the objective functions of DC-MIP, with $\boldsymbol{\pi}_i$ being the decision variables of DC-MIP (not the sum over variables of CS-MILP), we have:

$$\begin{aligned}
\sum_{s_i,a_i\in\mathcal{SA}_i} c_{i,k}(s_i \oplus a_i)\pi_i(s_i,a_i) &= \sum_{\widehat{s_i,a_i}\in\widehat{\mathcal{SA}_i}} c_{i,k}(\widehat{s_i,a_i}) \sum_{s_i,a_i\in\widehat{s_i,a_i}} \pi_i(s_i,a_i) \\
&= \sum_{\widehat{s_i,a_i}\in\widehat{\mathcal{SA}_i}} c_{i,k}(\widehat{s_i,a_i})\widehat{\pi}_i(\widehat{s_i,a_i}),
\end{aligned}$$

and we can then use the chain of equalities from the first part to get back to the objective

function of CS-MILP, where the $\widehat{\pi}_i(\widehat{s_i, a_i})$ are determined by the $\widehat{\pi}(\widehat{\boldsymbol{s}, \boldsymbol{a}})$, which in turn are determined by the equivalent solution to CS-MILP. Therefore every solution to DC-MIP has a solution in CS-MILP with the same objective value, and therefore the bi-implication holds. □

## B.4 Table of results

Table B.4.1 is discussed in Section 5.4.3.

| Problem | Times | Solutions | Problem | Times | Solutions |
|---|---|---|---|---|---|
| 1:5 | 1.04 | 91 | 4:9 | 0.86 | 90 |
| 1:6 | 1.44 | 122 | 4:10 | 1.66 | 115 |
| 1:7 | 2.06 | 141 | 4:11 | 2.0 | 128 |
| 1:8 | 2.84 | 180 | 4:12 | 3.09 | 176 |
| 1:9 | 3.15 | 188 | 4:13 | 3.71 | 216 |
| 1:10 | 3.52 | 198 | 4:14 | 3.63 | 217 |
| 1:11 | 3.76 | 201 | 5:9 | 0.61 | 69 |
| 1:12 | 4.46 | 237 | 5:10 | 0.95 | 90 |
| 1:13 | 4.94 | 255 | 5:11 | 1.44 | 112 |
| 1:14 | 5.22 | 258 | 5:12 | 2.57 | 157 |
| 2:6 | 0.89 | 84 | 5:13 | 3.01 | 195 |
| 2:7 | 1.12 | 107 | 5:14 | 3.29 | 198 |
| 2:8 | 2.03 | 156 | 6:10 | 0.66 | 66 |
| 2:9 | 3.03 | 165 | 6:11 | 0.88 | 86 |
| 2:10 | 3.19 | 175 | 6:12 | 1.89 | 128 |
| 2:11 | 3.31 | 180 | 6:13 | 2.45 | 176 |
| 2:12 | 3.79 | 220 | 6:14 | 2.97 | 179 |
| 2:13 | 4.27 | 237 | 7:11 | 0.51 | 61 |
| 2:14 | 4.55 | 242 | 7:12 | 1.16 | 108 |
| 3:7 | 0.75 | 78 | 7:13 | 1.95 | 153 |
| 3:8 | 1.52 | 133 | 7:14 | 2.31 | 161 |
| 3:9 | 2.06 | 144 | 8:12 | 0.91 | 95 |
| 3:10 | 2.8 | 158 | 8:13 | 1.43 | 140 |
| 3:11 | 2.93 | 161 | 8:14 | 2.23 | 153 |
| 3:12 | 3.75 | 204 | 9:13 | 0.47 | 87 |
| 3:13 | 4.02 | 234 | 9:14 | 0.89 | 97 |
| 3:14 | 4.13 | 237 | 10:14 | 0.6 | 80 |
| 4:8 | 0.58 | 81 | | | |

Table B.4.1: Heuristic results for large instances

# Bibliography


Adelgren, N. and Gupte, A. (2022). Branch-and-bound for biobjective mixed-integer linear programming. *INFORMS Journal on Computing*, 34(2):909–933.

Adusumilli, K. and Hasenbein, J. (2010). Dynamic admission and service rate control of a queue. *Queueing Systems*, 66:131–154.

Alderson, D. L., Brown, G. G., and Carlyle, W. M. (2015). Operational models of infrastructure resilience. *Risk Analysis*, 35(4):562–586.

Altman, E. (2021). *Constrained Markov Decision Processes*. Routledge.

Amari, S. V., McLaughlin, L., and Pham, H. (2006). Cost-effective condition-based maintenance using Markov decision processes. In *RAMS'06. Annual Reliability and Maintainability Symposium, 2006.*, pages 464–469. IEEE.

Anand, S. and Chidambaram, M. (1994). Optimal redundancy allocation of a PWR cooling loop using a multi-stage Monte Carlo method. *Microelectronics Reliability*, 34(4):741–745.

Aneja, Y. P. and Nair, K. P. K. (1979). Bicriteria transportation problem. *Management Science*, 25(1):73–78.

Argon, N. T., Ding, L., Glazebrook, K. D., and Ziya, S. (2009). Dynamic routing of customers with general delay costs in a multiserver queuing system. *Probability in the Engineering and Informational Sciences*, 23(2):175–203.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329.

Bei, X., Chatwattanasiri, N., Coit, D. W., and Zhu, X. (2017). Combined redundancy allocation and maintenance planning using a two-stage stochastic programming model for multiple component systems. *IEEE Transactions on Reliability*, 66(3):950–962.

Bei, X., Zhu, X., and Coit, D. W. (2019). A risk-averse stochastic program for integrated system design and preventive maintenance planning. *European Journal of Operational Research*, 276(2):536–548.

Benders, J. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numer. Math*, 4(1):238–252.

Bertsekas, D. (2012). *Dynamic Programming and Optimal Control: Volume I*. Athena Scientific.

Bertsekas, D. (2021). *Rollout, Policy Iteration, and Distributed Reinforcement Learning*. Athena Scientific.

Bertsimas, D. and Mišić, V. V. (2016). Decomposable Markov decision processes: A fluid optimization approach. *Operations Research*, 64(6):1537–1555.

Brown, S., Sinha, S., and Schaefer, A. J. (2024). Markov decision process design: A framework for integrating strategic and operational decisions. *Operations Research Letters*, 54:107090.

Chatterjee, K. (2007). Markov decision processes with multiple long-run average objectives. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 473–484. Springer.

Chen, D. and Trivedi, K. S. (2005). Optimization for condition-based maintenance with semi-Markov decision process. *Reliability Engineering & System Safety*, 90(1):25–29.

Chern, M.-S. (1992). On the computational complexity of reliability redundancy allocation in a series system. *Operations Research Letters*, 11(5):309–315.

Dai, J. G. and Shi, P. (2019). Inpatient overflow: An approximate dynamic programming approach. *Manufacturing & Service Operations Management*, 21(4):894–911.

Dantzig, G. B. and Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research*, 8(1):101–111.

Deep, A., Zhou, S., Veeramani, D., and Chen, Y. (2023). Partially observable Markov decision process-based optimal maintenance planning with time-dependent observations. *European Journal of Operational Research*, 311(2):533–544.

Dempe, S. (2002). *Foundations of Bilevel Programming*. Springer.

Devi, S., Garg, H., and Garg, D. (2023). A review of redundancy allocation problem for two decades: bibliometrics and future directions. *Artif Intell Rev*, 56:7457–7548.

Dimitrov, N. B., Moffett, A., Morton, D. P., and Sarkar, S. (2013). Selecting malaria interventions: A top-down approach. *Computers & Operations Research*, 40(9):2229–2240.

Dimitrov, N. B. and Morton, D. P. (2009). Combinatorial design of a stochastic Markov decision process. In *Operations Research and Cyber-Infrastructure*, pages 167–193. Springer.

Dowson, O., Gandibleux, X., and Kof, G. (2025). Multiobjectivealgorithms.jl: a julia package for solving multi-objective optimization problems.

Drexl, M. and Schneider, M. (2015). A survey of variants and extensions of the location-routing problem. *European Journal of Operational Research*, 241(2):283–308.

Dufour, F. and Piunovskiy, A. B. (2015). Impulsive Control for Continuous-time Markov Decision Processes. *Advances in Applied Probability*, 47(1):106–127.

Durrett, R. (2019). *Probability: Theory and Examples*, volume 49. Cambridge university press.

Ehrgott, M. (2005). *Multicriteria Optimization*. Springer Science & Business Media.

Florescu, I. (2014). *Probability and Stochastic Processes*. John Wiley & Sons.

Flynn, J. and Chung, C.-S. (2004). A heuristic algorithm for determining replacement policies in consecutive k-out-of-n systems. *Computers & Operations Research*, 31(8):1335–1348.

Fyffe, D. E., Hines, W. W., and Lee, N. K. (1968). System reliability allocation and a computational algorithm. *IEEE Transactions on Reliability*, 17(2):64–69.

Ghosh, M. K. (1990). Markov decision processes with multiple costs. *Operations Research Letters*, 9(4):257–260.

Guan, B., Li, Z., Coit, D. W., and Li, Y.-F. (2025). Review of the redundancy allocation problem to optimize system reliability. *Engineering Optimization*, 57(1):44–68.

Guo, C. and Liang, Z. (2022). A predictive Markov decision process for optimizing inspection and maintenance strategies of partially observable multi-state systems. *Reliability Engineering & System Safety*, 226:108683.

Guo, X. and Hernández-Lerma, O. (2009). *Continuous-time Markov Decision Processes: Theory and Applications*. Springer-Verlag.

Gurobi Optimization, LLC (2024). Gurobi Optimizer Reference Manual.

Kaparis, K. and Letchford, A. N. (2010). Separation Algorithms for 0-1 Knapsack Polytopes. *Mathematical Programming*, 124(1):69–91.

Karloff, H. (2008). *Linear Programming*. Springer Science & Business Media.

Kayedpour, F., Amiri, M., Rafizadeh, M., and Shahryari Nia, A. (2017). Multi-objective redundancy allocation problem for a system with repairable components considering instantaneous availability and strategy selection. *Reliability Engineering & System Safety*, 160:11–20.

Kemeny, J. and Snell, J. (1976). *Finite Markov Chains*. Springer-Verlag.

Keshavarz Ghorabaee, M., Amiri, M., and Azimi, P. (2015). Genetic algorithm for solving bi-objective redundancy allocation problem with k-out-of-n subsystems. *Applied Mathematical Modelling*, 39(20):6396–6409.

Kim, H. (2018). Maximization of system reliability with the consideration of component sequencing. *Reliability Engineering & System Safety*, 170:64–72.

Kim, H. and Kim, P. (2017). Reliability–redundancy allocation problem considering optimal redundancy strategy using parallel genetic algorithm. *Reliability Engineering & System Safety*, 159:153–160.

Konda, V. and Tsitsiklis, J. (1999). Actor-critic algorithms. *Advances in Neural Information Processing Systems*, 12:1008–1014.

Krishnan, K. R. (1990). Joining the right queue: a state-dependent decision rule. *IEE Transactions on Automatic Control*, 35:104–108.

Kulturel-Konak, S., Smith, A. E., and Coit, D. W. (2003). Efficiently solving the redundancy allocation problem using tabu search. *IIE Transactions*, 35(6):515–526.

Kuo, W. (2001). *Optimal Reliability Design: Fundamentals and Applications*. Cambridge university press.

Laporte, G. and Louveaux, F. V. (1993). The integer L-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters*, 13(3):133–142.

Lim, M., Daskin, M. S., Bassamboo, A., and Chopra, S. (2010). A facility reliability problem: Formulation, properties, and algorithm. *Naval Research Logistics*, 57(1):58–70.

Lim, M. K., Bassamboo, A., Chopra, S., and Daskin, M. S. (2012). Facility location decisions with random disruptions and imperfect estimation. *Manufacturing & Service Operations Management*, 15(2):239–249.

Ling, W. C., Andiappan, V., and Chew, I. M. L. (2021). Design of energy systems with redundancy allocation for unit operations based on supply reliability. *International Journal of Energy Research*, 45(15):21114–21139.

Lins, I. D. and Droguett, E. L. (2011). Redundancy allocation problems considering systems with imperfect repairs using multi-objective genetic algorithms and discrete event simulation. *Simulation Modelling Practice and Theory*, 19(1):362–381.

Lippman, S. A. (1975). Applying a new device in the optimization of exponential queuing systems. *Operations Research*, 23(4):687–710.

Liu, Z. (2022). *Optimizing Strategic Planning With Long-term Sequential Decision Making Under Uncertainty: A Decomposition Approach*. PhD thesis, University of Tennessee, Knoxville. Available at https://trace.tennessee.edu/utk_graddiss/7246.

Majety, S. R. V., Dawande, M., and Rajgopal, J. (1999). Optimal reliability allocation with discrete cost-reliability data for components. *Operations Research*, 47(6):899–906.

Meuleau, N., Hauskrecht, M., Kim, K.-E., Peshkin, L., Kaelbling, L. P., Dean, T. L., and Boutilier, C. (1998). Solving very large weakly coupled Markov decision processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 15, pages 165–172.

Morrison, D. R., Jacobson, S. H., Sauppe, J. J., and Sewell, E. C. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102.

Nunes, L. G. N., de Carvalho, S. V., and Rodrigues, R. d. C. M. (2009). Markov decision process applied to the control of hospital elective admissions. *Artificial Intelligence in Medicine*, 47(2):159–171.

Ozekici, S. (1988). Optimal periodic replacement of multicomponent reliability systems. *Operations Research*, 36(4):542–552.

O'Hanley, J. R., Scaparra, M. P., and García, S. (2013). Probability chains: A general linearization technique for modeling reliability in facility location and related problems. *European Journal of Operational Research*, 230(1):63–75.

Park, Y. W. (2020). MILP models for complex system reliability redundancy allocation with mixed components. *INFORMS Journal on Computing*, 32(3):600–619.

Parragh, S. N. and Tricoire, F. (2019). Branch-and-bound for Bi-objective Integer Programming. *INFORMS Journal on Computing*, 31(4):805–822.

Parragh, S. N., Tricoire, F., and Gutjahr, W. J. (2022). A branch-and-Benders-cut algorithm for a bi-objective stochastic facility location problem. *OR Spectrum*, 44(2):419–459.

Pillac, V., Gendreau, M., Guéret, C., and Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11.

Piunovskiy, A. (2004). Multicriteria Impulsive Control of Jump Markov Processes. *Mathematical Methods of Operations Research*, 60:125–144.

Piunovskiy, A. and Zhang, Y. (2025). Primal-dual programs for the constrained optimal impulse control: discounted model. *arXiv preprint arXiv:2504.18387*.

Powell, W. (2022). *Reinforcement Learning and Stochastic Optimization*, chapter 12, pages 653–699. John Wiley & Sons, Ltd.

Powell, W. B. (2011). *Approximate Dynamic Programming: Solving the Curses of Dimensionality, 2nd Edition*. Wiley & Sons, New Jersey.

Puterman, M. L. (2014). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Reihaneh, M., Abouei Ardakan, M., and Eskandarpour, M. (2022). An exact algorithm for the redundancy allocation problem with heterogeneous components under the mixed redundancy strategy. *European Journal of Operational Research*, 297(3):1112–1125.

Roijers, D. M., Vamplew, P., Whiteson, S., and Dazeley, R. (2013). A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113.

Ross, K. W. (1989). Randomized and past-dependent policies for Markov decision processes with multiple constraints. *Operations Research*, 37(3):474–477.

Serfozo, R. F. (1979). Technical note—An equivalence between continuous and discrete time Markov decision processes. *Operations Research*, 27(3):616–620.

Shang, K., Ishibuchi, H., He, L., and Pang, L. M. (2020). A survey on the hypervolume indicator in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 25(1):1–20.

Shone, R., Knight, V., and Harper, P. (2020). A conservative index heuristic for routing problems with multiple heterogeneous service facilities. *Mathematical Methods of Operations Research*, 92:511–543.

Süle, Z., Baumgartner, J., Dörgő, G., and Abonyi, J. (2019). P-graph-based multi-objective risk analysis and redundancy allocation in safety-critical energy systems. *Energy*, 179:989–1003.

Tavana, M., Khalili-Damghani, K., Di Caprio, D., and Oveisi, Z. (2018). An evolutionary computation approach to solving repairable multi-state multi-objective redundancy allocation problems. *Neural Computing and Applications*, 30:127–139.

Van Moffaert, K. and Nowé, A. (2014). Multi-objective reinforcement learning using sets of Pareto dominating policies. *The Journal of Machine Learning Research*, 15(1):3483–3512.

Van Noortwijk, J. M. (2009). A survey of the application of gamma processes in maintenance. *Reliability Engineering & System Safety*, 94(1):2–21.

Wakuta, K. (2000). Vector-valued Markov decision processes with average reward criterion: The multichain case. *Probability in the Engineering and Informational Sciences*, 14(4):533–548.

White, D. (1982). Multi-objective infinite-horizon discounted Markov decision processes. *Journal of Mathematical Analysis and Applications*, 89(2):639–647.

Whittle, P. (1988). Restless bandits: Activity allocation in a changing world. *Journal of Applied Probability*, 25(A):287–298.

Wolsey, L. A. (2020). *Integer Programming*. John Wiley & Sons.

Zhu, X., Bei, X., Chatwattanasiri, N., and Coit, D. W. (2018). Optimal system design

and sequential preventive maintenance under uncertain aperiodic-changing stresses. *IEEE Transactions on Reliability*, 67(3):907–919.

Zia, L. and Coit, D. W. (2010). Redundancy allocation for series-parallel systems using a column generation approach. *IEEE Transactions on Reliability*, 59(4):706–717.

Zoulfaghari, H., Zeinal Hamadani, A., and Abouei Ardakan, M. (2014). Bi-objective redundancy allocation problem for a system with mixed repairable and non-repairable components. *ISA Transactions*, 53(1):17–24.