On The Dynamic Allocation Of Assets Subject To Failure And Replenishment

Stephen Ford, B.A., M.Sc., M.Res.

Submitted for the degree of Doctor of Philosophy at

Lancaster University.

March 2021

STOR-i

excellence with impact

Abstract

Problems of the dynamic allocation of assets subject to both failure and replenishment are common. We consider a problem inspired by naval search, where unmanned aerial vehicles are required to search an area of ocean for targets. The vehicles will require refuelling or rearming; this is represented by the aspects of failure and replenishment. Similar models can arise from considering problems of search and rescue, environmental monitoring, or project management.

We formulate several versions of the problem, initially using the framework of a Markov decision process, bearing in mind trade-offs between real-world fidelity and mathematical tractability. We first consider models where rewards are gained independently from different tasks, before moving on to consider a specific kind of dependence in the rewards. We use a variety of mathematical techniques, including restless bandits, to formulate near-optimal policies for a slew of models.

We consider and investigate the various policies through comprehensive computational modelling. For the independent case, we find that a Whittle index policy is extremely close to optimal while being computationally efficient. For the dependent formulation,

we create a class of policies guaranteed to contain the optimal, parameterise the space, then choose the best from a limited set of parameters, augmenting with a single step of policy improvement.

We close with some thoughts about what we have learned, considerations about applying the results presented in this thesis, and a discussion of intensifications and extensions we did not have time to consider.

Acknowledgements

I would like to thank the following:

My supervisors Kevin Glazebrook, Peter Jacko and Michael Atkinson, for too many things to recount.

Chapter 2 appeared originally as a paper [1] in the European Journal of Operational Research; I would like to thank the referees and editors for their helpful suggestions.

My parents, for, amongst other things, a steady supply of cups of tea whilst I was writing up.

My brothers (they know what they did).

Declaration

I do 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.

Stephen Ford

Contents

List of Figures

List of Tables

List of Abbreviations

ADP	Approximate Dynamic Programming
CTMC	Continuous-Time Markov Chain
MDP	Markov Decision Problem
NMDP	Non-Markov Decision Problem
S&R	Search And Rescue
UAV	Unmanned Aerial Vehicle

List of Symbols

x	State of the system
e_i	The i^{th} basis vector with i^{th} element 1 and all other elements 0
a	An action
U, u	A policy specifying actions for each state
$p(x\|y)$	Probability of x given y
0, 1	vectors of all 0s and all 1s respectively
a, b, c	Arbitrary vectors
A, B, C	Arbitrary matrices
$\mathcal{A}, \mathcal{B}, \mathcal{C}$	Arbitrary sets
I	Identity matrix
N	Total number of assets
K	Number of reward-generating tasks
P	Transition probability matrix
Q	Transition rate matrix
B	Uniformisation constant
β	Discount factor
$\boldsymbol{\pi}$	Stationary distribution vector
ϕ	Relative value function

Chapter 1: Introduction

1.1 Describing the Problem

We shall begin by considering where our problem came from, why we care about it, and what kind of models we want to build, before then moving on to the details of the mathematical notation. It is easy to spend time creating increasingly complicated models, but we must ensure that the complexity accurately reflects the difficulties of the problem we are trying to solve.

There are three questions that we must keep in mind to prevent us from straying from the useful into the merely interesting. They are:

1. What is the problem that we are trying to solve?
2. What are the rules of the problem?
3. What is our plan for attacking the problem?

We begin this section by considering a variety of real-world applications which inspired

different parts of this thesis. We then move on to considering which kinds of models we want to build, as there are several places where trade-offs must be made between fidelity to the problem and mathematical or computational tractability. We focus throughout on the general shape of things more than the specific details, which we defer to subsequent chapters.

1.1.1 Real-World Inspirations

In this section, we first explain the meaning of the title of this thesis - "On The Dynamic Allocation of Assets Subject To Failure And Replenishment" - before then looking at four types of problem that inspired ours: naval search, search and rescue, environmental monitoring, and project management.

Our Problem: Dynamic Allocation With Failure And Replenishment

To explain what this title means, we now go through its significant words: dynamic, allocation, assets, failure and replenishment.

- **Dynamic**: We are concerned with problems that have an element of time or of change to them. This will lead us to consider controls that change with the situation and rewards that change as well. This is as opposed to static or once-and-for-all problems where we apply a single decision and get a single reward.

- **Allocation**: We wish to understand not only the dynamics of the systems but also how best to control them. We assume that there are some actions we can take to affect the system and that we wish to maximise some changing reward gained from the system, or equivalently to minimise some loss. Our controls will take the form of some kind of allocation of assets, generally to 'tasks' in the system.

- **Assets**: Our allocations will be of some assets in the system. They will generally be discrete, for both mathematical and real-world reasons: drones, ships and dogs come in whole numbers. We can reasonably assume that as we are the one in control of them, we know their properties, even if the eventual lifetimes are random, and that any lack of knowledge will be about the system. We might not know exactly when they will fail or be repaired, but will know that they can fail or be repaired.

- **Failure**: Our assets will have finite lifetimes, after which they fail; were the lifetimes infinite, we would have a static optimisation problem. We will consider a variety of possibilities for the forms of the failure: in some cases we might know the lifetimes exactly, whereas in others we might start off with no idea, and have to infer it from the observed results. If the lifetimes are unknown, then we can model them as random, and we may or may not know the parameters for the random distributions exactly. This ties into the first point: both the system and our controls must be dynamic, to capture the element of failure.

- **Replenishment**: If assets only ever failed, we would have a very short problem

indeed. There must be some process of replenishment, which returns the assets to us, ready to reallocate. This makes the system as a whole recurrent, an important aspect we need to include in our mathematical models. As with the failures, the repair times might be deterministic or random, have known or unknown parameters, and could depend on the number of assets being repaired at the time.

Taken together, the key aspects are optimisation and cyclic choices. We want to optimise our allocations, and the assets go through a regular cycle of allocation, failure and repair. One other thing worth stressing is randomness: we assume that there is some stochastic element to the system, whether inherently in the dynamics or arising from our ignorance.

In the next four sections, we are going to talk about the major real-world scenarios that lead us to this approach and title, to clarify our motivations.

Naval Search

This is our first and most important application, which should be borne in mind when reading the rest of this thesis. The best way to explain this is to quote from the proposal document for this PhD:

"Consider our assets to be UAVs searching for targets in a communication-degraded environment. Some UAVs search for targets (task 1 is search, say) while other UAVs form a line-of-sight communication network (tasks $2, \ldots, K$ *are communication at*

distinct network locations, say) back to the centre of operations (base). A target is only detected if its sighting is reported back to base successfully. In order for this system to function effectively, we need some UAVs searching, with enough UAVs also deployed to form a fully connected communications network from the search region to base. UAVs have a limited endurance and must periodically return to refuel. After refuelling, a UAV is sent back to the area of operation as either a searcher (task 1) or a communicator (one of tasks $2, \ldots, K$*). The goal is to determine effective approaches to UAV deployment."*

Problems of UAV utilisation, especially for searching, have been much studied in the literature*, but our particular interest is the use of UAVs in naval search problems. This has several particular features we wish to reflect in any mathematical formulations we use.

Firstly, the oceans are unhelpfully large [8]. This seemingly obvious fact is in fact significant: it is the origin of the 'failure and replenishment' part of our title. This is because unlike in other applications, we cannot necessarily expect to search all of the target area in a single pass, at least to a sufficient level of accuracy. Even if our assets are comparatively long-life in terms of fuel or charge, we probably have to contend with mechanical failures and the like.

Secondly, we would expect our assets to be discrete whether they are UAVs, planes, ships, or submarines, and quite probably homogeneous. By homogeneous we do not necessarily mean identical, just that they are similar enough that the differences can

*For a review, see [2] and [3, 4, 5, 6, 7] for some particular examples.

be folded into the randomness present in the system.

Continuous assets can sometimes be better-behaved mathematically, even when reality is discrete - this is called fluid approximation [9, 10, 11] - but under our circumstances, if we make the system continuous then it will no longer be dynamic. It would instead stay in a steady state, depending on the control applied.

Lastly, we would expect to have a significant number of assets: more than one or two, but probably less than a hundred. An important reason for the use of UAVs as opposed to traditional manned aircraft is that they are cheaper [12] and can be used *en masse*. This is precisely why we seek to find 'effective' and not necessarily optimal approaches to UAV deployment: the numbers involved in our problems may simply be too large to achieve anything more.

We therefore have to bear in mind the computational complexity of any calculations our allocation decision processes require. We are unlikely to require allocations to be made each second - timescales of hours are more likely - but we still need to ensure we can calculate what decision needs to be made without causing undue delay to the allocation process.

The number of assets is another reason to favour a discrete model over a continuous one. Continuous approximations naturally become more accurate as the number of assets increases, and our numbers are likely small enough that the errors would be significant - there is no way to search with half a drone. That being said, it is always possible to deal with rounding by randomisation, so that if the policy dictates that

we allocate 0.6 of a drone, we allocate a single drone with probability 0.6.

Search and Rescue

Search and rescue operations (hereafter S&R) are often retrofitted onto existing methods or equipment as a post hoc justification [13], but one common reason to search large areas of otherwise featureless ocean is to locate boats or ships that are in trouble. Similarly, we might want to search moors or mountains to find lost hikers, or to search through buildings collapsed by an earthquake to find any survivors.

Each of these cases has its own difficulties, but two peculiarities are generally common in S&R problems: finite horizons, and inhomogenous assets [14]. In a S&R problem, we often have a fixed number of targets to find, with the problem being to minimise the expected time until we have found all of them. This encourages us to be more myopic in our allocation of assets: if a failure happens after the last target is found, we have no reason to care. This is an aspect that is easily incorporated into mathematical models, but requires us to have information about how long we might have to find our targets.

Similarly, the rewards might be 'impatient' [15] because if we delay the targets could die.

The second aspect is equally important, but more mathematically difficult to incorporate. S&R operations, being time-limited, are often carried out with whatever assets are available at that moment. As such, the assets will not necessarily all be the same,

but might be of several different classes. In any formulation this would increase the number of possibilities we have to consider, but this complication may be necessary.

Environmental Monitoring

At the other end of the spectrum, we have problems of long-term monitoring. This is another common use-case for UAVs [16, 17, 18, 19], which are cheap enough to be worth using for mundane monitoring. A simple example would be monitoring a forest looking for signs of disease: here the problem is so long-term as to be effectively infinite-horizon and so considerations of failure are unavoidable. We would want to maximise the long-term average reward, which could be a statistic such as average area covered, or length of time between checks.

Similar but slightly different is monitoring wildlife [20]. Even if using static cameras, they can break and have to be replaced, and we would have limited resources to replace them. Here each 'task' would be one location where we could put a camera, and we would have fewer assets than tasks. Our objective might be to get as accurate a count as possible of the population present; we would seek not just to record as many animals as possible, but also to do so in different locations.

The monitoring of a forest fire [21] would also be long-term, as such fires can burn for weeks on end. Here we might have a trade-off between risk and reward: the closer to the front of the fire we send our assets, the more likely they would be to fail. In addition, we might have multiple failure modes: crashes due to smoke and ashes, and

ordinary failures caused by running out of fuel.

Project Management

Consider a problem where we have a project to manage, with assets as employees or contractors, and a manager needing to assign them to projects composed of discrete tasks. People can only work for so long before needing a break, a rest or taking leave, but such features are typically ignored in the project management literature [22]. We might even have multiple stages of failure, with workers getting tired and their performance degrading long before they require a break.

Mathematically, this is different because there might be significantly more assets than tasks, with decisions being made on a scale of weeks or months [23]. As such, computational efficiency would be less of a concern; however, we might have much more uncertainty about completion times and failure times, and several different failure modes.

1.1.2 Formulating Models of our Problem

We have now given a generic description of our problem, as well as an overview of the real-world applications whose behaviour we want to capture. As explained in Section 1.1.1 above, the essence of our problem is the repeated allocation of assets to a finite number of discrete tasks. In this section, we explore several aspects of turning this vague description into a concrete model, and illustrate with suitable diagrams.

This section should be read with an eye on Section 1.1.1: we want to make choices which reflect the difficulties outlined above.

The Most General Model Necessary

There are many choices we have to make in formulating a model for our problem. We therefore start off by creating a very general model for our problem. There are five key aspects:

1. Assets arrive into the system according to some process. This could depend on the state of the system (i.e. repairs) or be entirely external. The assets might be hetereogeneous or all the same.

2. We allocate these assets to tasks, of which there are a finite number*. The tasks are of a fixed number, and remain the same throughout, with neither impatience nor expiration on completion. We might also choose not to allocate and to reject the incoming assets from the system entirely, or instead to reserve arriving assets for future allocation.

3. We gain reward according to some process depending on the number of assets at the tasks. This might be random or deterministic, and the gain process might be completely understood or initially unknown. The rewards are considered to be stationary in the sense that the same allocation will always give the same

*Even if there are an infinite number of tasks, if the system is stationary, which we generally assume, then as we only have a finite number of assets only a finite number of tasks will ever be used.

reward.

4. The assets stay allocated until they fail, and leave the tasks; this might be a true failure, or a predetermined decision to withdraw them. The assets may leave the system entirely, or go to a repair process, or one of several repair processes.

5. We seek to maximise the total overall reward: this might be up to some finite deadline, accrued in a discounted manner, or the infinite-horizon long term average.

We have deliberately left most of the details unspecified, and will specify them in subsequent sections. Nevertheless, if we look at Figure 1.1.1, the essential flow of the problem is clear, and it is the flow that is our main concern.

It is worth dicussing here that there are three slightly different things we want to be able to do with any model: stochastic modelling, performance evaluation, and optimisation. Stochastic modelling means simply to formulate the model - to be able to describe it in a complete and comprehensible fashion, stating the uncertainty and randomness present.

Performance evaluation means to be able to understand the dynamics, either analytically or through simulation. It is perfectly possible to formulate a sensible model that we can neither solve analytically nor simulate on a computer, usually due to the model's size. For our problem, understanding the dynamics at its simplest would mean applying a very simple policy, and seeking to answer questions such as 'How many assets are in the system on average?' or 'What's the lowest number of assets

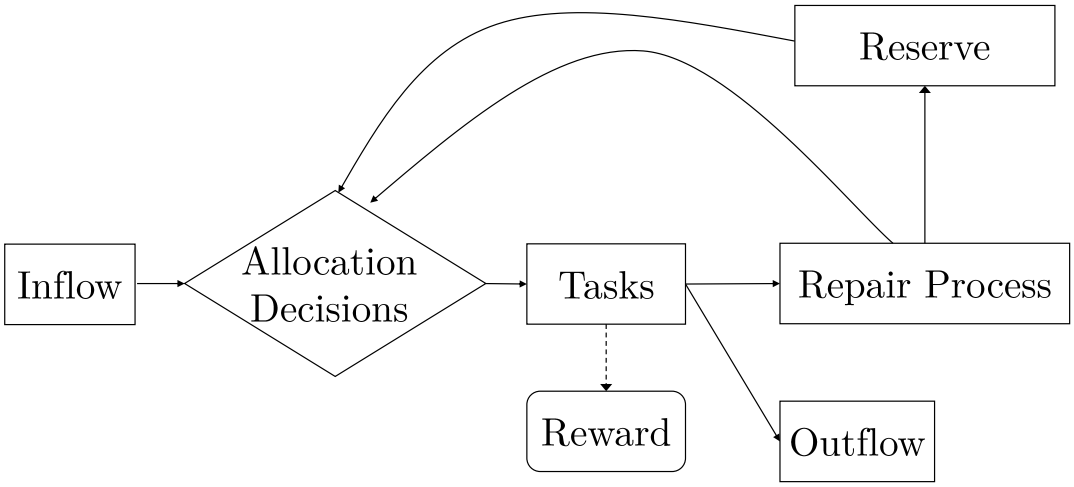


Figure 1.1.1: Flowchart detailing the generic dynamics of our problem. Inflow and Outflow refer to the assets, which may be allowed to enter or leave the system. Solid lines indicate flow of assets, whilst dotted line indicates rewards.

we ever get at task 2?'.

The third and most challenging aspect is optimisation: to specifically design policies that work better than others. To be able to do this we need to be able to get some grasp on the dynamics, but also to have an idea of what we want to achieve. In this section we focus on the first aspect, while Section 1.2.1 looks mainly at the second; it is worth bearing in mind that our ultimate goal is the third aspect, the optimisation of the real-world systems concerned.

Closed or Open?

The first detail we tackle is a combination of the first and fourth parts given above: is the system closed, and so the assets cycle round, or is it open, with assets arriving to

and departing from the system. There are two different questions we need to consider: which is more realistic, and which is more tractable.

To answer the first question: it depends. More usefully, it depends on the application but closed models will usually be more accurate. Particularly in the all-important naval search application we would expect to be handed a box of assets and told to get on with it, and so we would be working under a closed model.

Looking at the other applications listed in Section 1.1.1, search and rescue and environmental monitoring lean towards closed - a fixed set of assets - while project management would probably be open, with employees cycling in and out of the system. That being said, there are always choices in the modelling: with project management, we could always treat job roles as our assets, and replacing an employee by another employee with the same job as a repair, giving a closed system.

In terms of tractability, an open system tends to give an infinite state space. Even though in practice we are not going to have models where the number of assets increases without bound, we must still consider the possibility if we choose to model a system as open. This is counterintuitively often nicer from a mathematical standpoint, as we don't have to consider edge effects arising from an imposed upper bound.

We illustrate the difference between a closed and an open system in Figure 1.1.2; when looking at the open system remember that incoming assets can be rejected, so the inflow is more a flow of opportunities to get assets than of assets which we must receive.

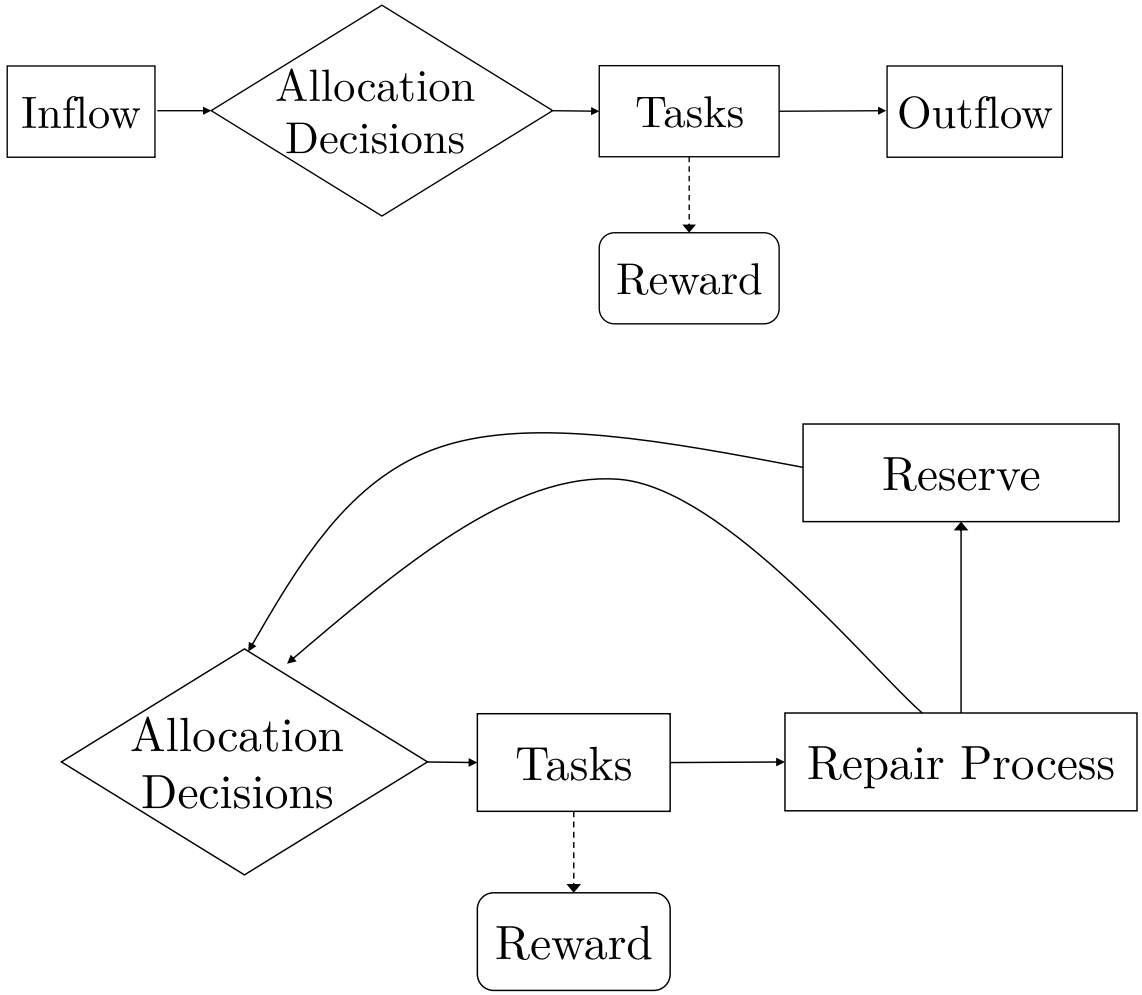


Figure 1.1.2: Systems with open (top) and closed (bottom) dynamics.

Dependent or Independent?

For our purposes, dependent and independent are defined as follows: a reward function on variables $x_1, \ldots, x_n$ is independent if it can be written in the form $\sum_{i=1}^{n} g_i(x_i)$, and is dependent otherwise. Elsewhere in the literature this form may be called additive; the effect is the same.

Why is this difference significant? As outlined in Section 1.1.1, the problem that started this off had communications with search as a key part of the difficulty of the problem. This is an inherently dependent structure: getting reward depends on searching and communicating at the same time.

The difference becomes important later on when we introduce the restless bandit framework in Section 1.2.2. This is only applicable if we can disentangle the different tasks from each other, which is only possible if the rewards are independent.

For the dependent case, we have no such framework and so the problem will prove much harder. This will prove a theme throughout this thesis: if there are two options, the more realistic one is inevitably harder.

Markovian, Phased, or General?

We discuss Markov chains later, in Section 1.2.1, but for now the key word is memoryless: we don't need to keep track of how long things have been at a task.

This is naturally helpful: the state of the system is then entirely summarised by the locations of the assets, with no other information such as ages required. Unless we are dealing with radioactive decay very few processes in real life are actually Markovian, but the mathematical helpfulness is so great that we will always start with this assumption.

Slightly better are Erlang models. Here, an asset's dynamics are described by a number of phases, each individually Markovian with the same rate. Now all we need to keep track of is where each asset is, and which stage it is currently in. This is more accurate, and good for modelling dynamics that genuinely do have multiple phases, but a little less tractable.

A fully general model allows us to pick arbitrary distributions for how long events take to happen. The consequences of this are obvious: we can accurately represent any behaviour, but have to account for any behaviour. This lends itself to ad-hoc approaches tuned to the distributions in question.

We illustrate the different dynamics of a task in Figure 1.1.3.

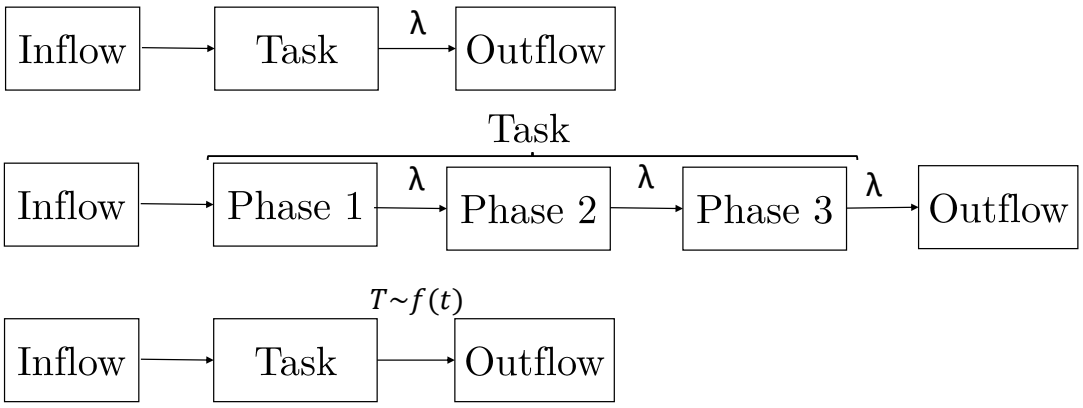


Figure 1.1.3: Dynamics for a single asset going through a single task with Markovian (top), Erlang (middle), or general (bottom) dynamics. The λ indicates that flow happens at a rate λ, so with exponential distribution. The expression $T \sim f(t)$ means that the time T for a failure happens according to some pdf f.

1.2 Mathematical Review

Now that we have an idea of what kinds of models we want to create, we can describe the mathematical background necessary for these models. This is mostly standard material, and so we are mainly concerned with laying out our notation and with a few unusual complexities to do with computation and impulsive controls. We start by explaining Markov Decision Processes and then move onto bandit problems, including restless bandits.

It is worth stressing that these techniques are in the main well established* and very general. Dynamic programming and the Bellman equation are a generic and powerful approach that can be successfully applied in many fields, but the application is very

*Except for the detail on impulsive controls.

often the hard part.

1.2.1 Markov Decision Problems

The framework for most of our mathematical formulations will be that of a Markov Decision Problem (MDP). Throughout this section, we will largely be following [24], which is a well-written textbook covering MDPs. These definitions are standard, but the notation used does vary significantly between works; the notation chosen here is set up to be convenient for the applications we use it for.

We have chosen the framework of MDPs more for their tractability than for their realism. There is in mathematical modelling work an eternal tension between reality and tractability, where the real-world models can't have anything useful done with them, and the solvable models don't apply to anything that actually exists. MDPs are more over to the 'tractability' side of the equation, and it seemed sensible to start with models we can work with and then to try to add complications on from there.

We should briefly mention measure theory. As our problems are in discrete spaces and derived from real-world problems, everything necessarily is well-behaved. We therefore will not discuss measure theory after this.

Markov Chains

We start with the 'M' in MDP. We write Markov chains as follows: we have a system of some sort, which has a known state x in a statespace $\mathcal{X}$. This evolves in simple discrete time $t = 0, 1, \ldots$, possibly with a maximum time, possibly in infinite time. At each timestep, the next state is determined from the current one according to a (possibly infinite) matrix $P(t)$, with:

$$p(x(t+1) = y | x(t) = x) = P(t)[x, y] \tag{1.2.1}$$

The system evolves in discrete jumps, going from one state to another, inside the set $\mathcal{X}$.

For a continuous-time Markov Chain (CTMC) [24, Chapter 11], with time $t \in [0, \infty)$, we instead have the transition rate matrix Q, with $Q[x, y]$ the rate at which we transition from state x to state y. The length of time before a transition when the system is in state x is exponentially-distributed with rate parameter $-Q[x, x]$; the probability that we transition to state y is $Q[x, y]/(-Q[x, x])$.

Now for some definitions. These are standard, but worth listing, as we will use them throughout this thesis.

Definition: If $P(t)$ does not depend on t, we say that the Markov chain is stationary.

Definition: If $\mathcal{X}$ is finite, then we say the corresponding Markov chain is finite.

Otherwise the Markov chain is infinite.

Definition: A Markov chain is irreducible if we can go from any state to any other state with non-zero probability, given enough time: $\forall x, y \in \mathcal{X}, \exists t$ such that $p(x(t) = y|x(0) = x) > 0$.

Definition: A state is aperiodic if the greatest common divisor of all number of steps which it can take to return to the state is 1:

$$g.c.d.(\{t \in \mathbb{N} \text{ such that } p(x(t) = x|x(0) = x\}) = 1 \tag{1.2.2}$$

Definition: A Markov chain is aperiodic if all its states are aperiodic.

Definition: A state x is transient if, starting from x, there is a non-zero probability that we never return to x: $p(x(t) \neq x \, \forall \, t \in \mathbb{N}|x(0) = x) > 0$.

Definition: A state is recurrent if it is not transient.

Definition: A Markov chain is recurrent if all its states are recurrent.

Definition: A state x is positive recurrent if starting from x, the expected time to return to that state is finite: defining $T_x = \min(t \in \mathbb{N}_{>0}|x(t) = x)$ to be the first return time, then this is equivalent to $\mathbb{E}[T_x] < \infty$.

Definition: A Markov chain is positive recurrent if all its states are positive recurrent.

Definition: A Markov chain which is stationary, irreducible, and positive recurrent

is called **ergodic**.

Note that while states can be recurrent but not positive recurrent - this is called being null recurrent - for stationary Markov chains this can only happen if they are infinite, and so we will not have to consider this difficulty, as our chains will be both stationary and finite.

Also, both aperiodicity and recurrence are class properties: if a Markov chain is irreducible, than any state is aperiodic or recurrent iff all states are.

Uniformisation

If the transition rates $Q[x, y]$ of a CTMC are bounded above by some value B, we can transform a CTMC with transition-rate matrix Q into an equivalent discrete-time model [24, Chapter 11]. What we do is make the system always transition at a constant rate B, but add a new 'dummy' transition that does not change the state; it occurs with probability $(B + Q[x, x])/B$, remembering that $Q[x, x] < 0$ always.

The resultant transition matrix then has $P[x, y] = Q[x, y]/B$ for $x \neq y$, and $P[x, x] = (B + Q[x, x])/B$. The sum of each row is then 1, and all of the elements are non-negative, so that this is a proper stochastic matrix. This procedure is called uniformisation, because it makes all of the transition rates the same - uniform.

If a Markov chain is finite, then the transition rates are bounded above as long as none of them are infinite, and so we can always uniformise such a chain. This may be

inefficient if there is a large gap between the largest transition rate and the majority of them, but in practice we have greater concerns.

Finding the Stationary Distribution

Suppose we have a finite-statespace discrete-time Markov chain, or equivalently a finite uniformised CTMC. If it is ergodic, then the stationary distribution $\boldsymbol{\pi}$ exists and is given by [24, Appendix A, section 4]:

$$\boldsymbol{\pi}^T P = \boldsymbol{\pi} \tag{1.2.3}$$

normalised so that the components of $\boldsymbol{\pi}$ sum to 1.

Suppose we have an ergodic Markov chain. How do we actually calculate $\boldsymbol{\pi}$ on a computer? While the equation $\boldsymbol{\pi}^T P = \boldsymbol{\pi}$ seems simple, in practice the first difficulty is mapping $\mathcal{X}$ to the natural numbers so that a computer can work with it. Sometimes the statespace may in fact be a set $\{1, 2, \ldots, n\}$ but more often we have to be clever. The numbering system does not matter: the results are the same regardless of the ordering of the states, but we still need to produce an enumeration of the states; see Appendix B.4 for how we do this in practice.

Additionally the system given in Equation (1.2.3) is degenerate. The easiest way to show this is to rearrange the system to $\boldsymbol{\pi}^T(P - I) = \mathbf{0}$, by moving the column vector $\boldsymbol{\pi}$ to the left-hand-side. The sum of each row of P is 1 by definition, but we then

subtract one off from each row, from the identity matrix I, so that each row sums to zero. As such, if we take a vector $\mathbf{1} = (1, \ldots, 1)$, then $\mathbf{1}^T(P - I) = \mathbf{0}$, so the matrix $P - I$ has at least one left eigenvalue of value 0, so the system is degenerate.

This degeneracy makes sense: we have to normalise the solution $\boldsymbol{\pi}$ to sum to one to get a unique answer. There are two ways to deal with this. The first is to add another row to $P - I$, with the row entirely composed of ones, and replace the zero vector on the LHS by a vector $(0, 0, \ldots, 0, 1)$ of length $|\mathcal{X}| + 1$ where all elements except the last are equal to zero. This is more than a little ugly, and means that the matrix we are dealing with is no longer square.

Alternatively, we can take advantage of the degeneracy, combined with the fact that we are working on a computer, which does not have infinite precision. Consider the system $\boldsymbol{\pi}(P - I) = \mathbf{0}$. This can be equivalently written $\boldsymbol{\pi}(P - I) = 0\boldsymbol{\pi}$ where the 0 on the LHS is a scalar not a vector. This is an eigenvalue problem, where we need to find the left eigenvector of the matrix $P - I$ corresponding to the eigenvalue 0.

We can now exploit the computer's lack of precision: there are standard routines* for finding the eigenvalue of least absolute magnitude, and finding the corresponding eigenvector. So all we have to do is take that eigenvector, normalise it to sum to 1, and we have $\boldsymbol{\pi}$.

To summarise how we calculate $\boldsymbol{\pi}$:

1. Form the matrix $P - I$.

*My code is written entirely in R, and I use the RSpectra [25] package.

2. For that matrix, find the left eigenvector corresponding to the eigenvalue of least absolute magnitude.

3. Normalise that eigenvector so that it sums to 1.

For the details of how we implement this, see Appendix B.2.

In general, for a finite Markov chain whose statespace has size $|\mathcal{X}|$, the runtime to find its stationary distribution $\boldsymbol{\pi}$, assuming it exists, is of $O(|\mathcal{X}|^3)$. We are more concerned with the entirely implementation-dependent real-world time taken to actually find $\boldsymbol{\pi}$. It is therefore incumbent upon us to formulate the model so as to keep $\mathcal{X}$ small, as that is what the runtime depends on: if $|\mathcal{X}|$ is large then the real-world runtime can be prohibitive.

Decision Processes & Policies

We have explained the 'Markov' part; now for 'Decision Problem'. A Markov Decision Problem (MDP) is a Markov chain where we can control the transition probabilities to some degree, and we get some kind of reward, depending on the state and control applied. We assume the Markov chain is ergodic, stationary*, and finite, with notation as laid out above.

Here is what makes a MDP: for each state x, there is a space of possible actions $\mathcal{A}(x)$; at each discrete timestep $t = 0, 1, \ldots$ we choose a single action $a(t) \in \mathcal{A}(x(t))$. This gives transition probabilities $p(x(t+1) = y | x(t) = x) = P(x, y | a(t))$.

*Under a stationary policy, a term to be defined in a moment.

We focus for now on stationary deterministic policies: stationary, in that they do not depend upon time, and deterministic, so they always make the same decision in the same state. The first condition is entirely reasonable as we have already said that the chain is stationary, and make the rewards time-invariant as well. If everything about the system remains the same, our decision making process - but not the action determined by our decision making process - should remain the same.

The second condition is not automatically true, but our problem is finite so that there is always a best decision, and so any randomness in the decision making process should arise from our ignorance of which is the best decision to make. There might also be cases where two possible decisions are equally good, but in that case we can just pick one, and always make the same decision for that set of circumstances.

Under these constraints we can specify our actions by a policy u, which is a map $u : \mathcal{X} \to \mathcal{A}(\mathcal{X})$. These constraints might seem harsh, but if an optimal (in a sense we define in a moment) policy exists for a given MDP whose associated Markov chain is stationary*, then there is also one which is stationary and deterministic [24, Chapter 5].

Now for rewards: if we take an action a in state x, we get a reward $R(x, a)$. We assume that the rewards are deterministic, because otherwise we can just use the expected reward, and that they are stationary. We also assume that the reward function is not exceptionally complex to calculate, so we do not need to worry about being efficient in our usage of evaluations of it.

*As long as a stationary policy is applied, obviously.

A policy is called optimal if it maximises the total expected reward gained. This is a natural definition, but maximises over what? There are three main possibilities:

1. The finite-horizon reward [24, Chapter 4]: $\mathbb{E}[\sum_{t=0}^{t=T} R(x(t), a(t)]$ for some $T \in \mathbb{N}$.
2. The discounted infinite-horizon reward [24, Chapter 6]: $\mathbb{E}[\sum_{t=0}^{\infty} \beta^t R(x(t), a(t))]$ for some discount rate $\beta \in (0, 1)$.
3. The infinite-horizon time-averaged reward [24, Chapter 5]:

$$\liminf_{H\to\infty} \frac{1}{H} \mathbb{E}\left[\sum_{t=0}^{H} R(x(t), a(t))\right] \tag{1.2.4}$$

All of these expectations are taken with respect to a given policy u, so that when in a state x we take the action $a(t) = u(x(t))$, which is well defined because we require that u is both stationary and deterministic.

We are going to use the last of these, mainly because if the stationary distribution $\boldsymbol{\pi}$ exists, then there is a much neater expression for the average reward. Define $\boldsymbol{\pi}^u$ as the stationary distribution of the Markov chain under the policy u. Then the expected long-term average reward is simply:

$$\sum_{x\in\mathcal{X}} \boldsymbol{\pi}^u(x) R(x, u(x)) \tag{1.2.5}$$

This is no harder to calculate* than $\boldsymbol{\pi}^u$, and therefore can be calculated in $O(|\mathcal{X}|^3)$

*Remember that we assumed that $R(x, a)$ is never severely difficult to evaluate.

time, which will be handy later for evaluating policies.

Given this, we say a policy u is optimal if it maximises the reward gained, over the set of all possible policies $\mathcal{U}$.

Dynamic Programming

We can evaluate the reward gained by policies, and we now know what it means for a policy to be optimal, but can we actually find an optimal policy? The first question is: does an optimal policy exist? One does for the problems we will consider because the problems are finite in both $\mathcal{X}$ and $\mathcal{A}(x)$ and so there are only a finite number of possible policies*. Given this we can find an optimal policy, and can do so both deterministically and in finite time [24, section 5.4]. In this section we consider only the infinite-horizon time-averaged reward, but the same approach works for discounted infinite-horizon.

We start by reducing the problem to one we can solve, by shrinking the time-horizon. If we only had one timestep to worry about, then the optimal policy would be to pick the a that maximises $R(x, a)$. If we had two timesteps, we would want to maximise $R(x, a) + \sum_{y \in \mathcal{X}} P(x, y|a)R(y, a(y))$: the reward we get at this timestep, plus the expected reward we get at the next timestep. We can continue this onwards until infinity, getting a linear algebra problem.

The resultant set of equations is called the Bellman equation. For each state $x \in \mathcal{X}$,

*Because as mentioned above there is guaranteed to be an optimal policy which is both stationary and deterministic, for the infinite-horizon average reward case [24, Chapter 5].

it is as follows:

$$V + \phi(x) = \max_{a \in \mathcal{A}(x)} (R(x,a) + \sum_{y \in \mathcal{X}} P(x,y|a)\phi(y)) \tag{1.2.6}$$

Here V is the long-term average reward, and $\phi(x)$ is called the relative value function: $\phi(x) - \phi(y)$ expresses how much we would prefer to be in state x than to be in state y. As there is one of these equations for each state, the problem is underdetermined. We have $|\mathcal{X}|$ equations, and $|\mathcal{X}| + 1$ parameters: the $\phi(x)$ and V. Therefore we have to pick some reference state x_0 and specify $\phi(x_0)$ arbitrarily.

The remarkable fact is, that if we can find some $a(x)$ and $\phi(x)$ such that the max on the RHS of Equation (1.2.6) is attained by $a(x)$ then the resultant policy is optimal [24, section 5.5]. The key conditions that we need for this are that everything is stationary and finite, and that the rewards are bounded in absolute value.

Policy Improvement

How do we actually find an optimal policy? The answer is to iterate [24, section 6.4], much as we did to generate the Bellman equation Equation (1.2.6). We proceed as follows: first pick some policy u_0 and relative value function ϕ_0, which are largely arbitrary, but for computational reasons should induce a recurrent chain and be of reasonable absolute magnitude respectively. We then do the following, iterating discretely over $n = 0, 1, 2, \ldots$:

1. For each $x \in \mathcal{X}$, set $u_{n+1}(x) = \arg\max_{a \in \mathcal{A}(x)}(R(x,a) + \sum_{y \in \mathcal{X}} P(x,y|a)\phi_n(y))$.

2. If $u_{n+1} \equiv u_n$ so that they prescribe the same actions for all states, stop and return u_{n+1}.

3. Form the system of linear equations given by $V + \phi_{n+1}(x) = (R(x, u_{n+1}(x)) + \sum_{y \in \mathcal{X}} P(x,y|u_{n+1}(x))\phi_{n+1}(y)$ for each $x \in \mathcal{X}$ and $\phi_{n+1}(x_0) = 0$.

4. Solve this system of linear equations for V and ϕ_{n+1}.

5. Increment n by 1, and return to step 1.

Notably, each iteration of the algorithm, known as policy improvement* results in the average reward V increasing. Additionally policy improvement is guaranteed to terminate in at most $2^{|\mathcal{X}|}$ iterations [26], and in practice it takes much fewer; for the problems we consider generally less than 10, despite $|\mathcal{X}|$ ranging from 100 to 1000.

Because each iteration is guaranteed to increase V, we can run a single step or a small number of steps of policy improvement on existing policies, and be certain that we are getting a better one. This might seem obvious, but it is actually quite powerful: we can consider the problem, come up with a policy that is for various reasons going to perform quite well, and then run policy improvement on it for just one step.

The better the policy we start with, the better the results will be in practice, and this can be much quicker than running policy improvement to completion, which, as mentioned above, can take extremely long. Additionally, the first step usually gives

*Some descriptions call it policy iteration.

the greatest improvement in practice.

One last point: for computational reasons, it is advisable to set a hard cap on how large the number of iterations n can be, and to simply terminate and return u_n if we reach this. We do so, generally putting the cap around 50, which is high enough that it is almost never reached. For the details of how we have implemented this, see Appendix B.1.

Impulse Controls

In some circumstances controls may occur instantaneously with regard to the process [27]. That is, when we apply a control a instead of altering the transition probabilities $p(x, y|a)$, we jump deterministically to a new state $x \to x + a$. The control $a = 0$ therefore corresponds to not applying a control at that instant. This might seem entirely contrary to the nature of MDPs, but since the transitions between states are instant in a standard Markov chain, this can still be incorporated within the existing framework.

Thinking about what happens when we apply an ordinary control, we alter two things: the reward rate $R(x, a)$ and the transition probabilities $p(x, y|a)$. When we apply an instantaneous control, the reward rate afterwards is $R(x + a, a)$, while the transition rates become $p(x + a, y|a)$. Therefore, we can treat impulsive controls exactly like regular controls, by changing the Bellman equation Equation (1.2.6) to the following:

$$V + \phi(x) = \max_{a \in \mathcal{A}(x)} (R(x + a, a) + \sum_{y \in \mathcal{X}} P(x + a, y|a)\phi(y)) \tag{1.2.7}$$

This lets us incorporate sudden-change controls in the standard MDP framework, which will come in useful in later chapters. We do have to be a bit careful when programming this up: we need to make sure that we never want to apply two controls, one immediately after the other.

What we need to do is make sure that for any state x, we apply a control $a(x)$ such that the control applied in the new state $a(x + a) = 0$; we need this to avoid visiting states but spending zero time there, which causes some interesting numerical issues. This can be done by always taking the maximal optimal control: the optimal control that moves as many assets as possible.

How does this differ from regular controls? The essence is: regular controls alter the transition probabilities $p(x, y|a)$, while impulse controls alter the state $x \to x + a$. Regular controls are more appropriate for circumstances where we are directing the flow of the system, whereas impulse controls are better when we impose changes, and let the system evolve on its own between applications of controls.

We can apply policy improvement to Equation (1.2.7) just as we could to Equation (1.2.6): all that changes is the system of linear equations. The resultant system is only slightly different, and can be solved in the same way.

1.2.2 Multi-Armed Bandits

Now that we have the generic notation for Markov chains and MDPs in place, we can now describe a specific kind of decision problem, known as a multi-armed bandit problem. We first talk about the standard case, including how to formulate an optimal policy, before going on to describe restless bandits, which are much more applicable to our problems.

The Multi-Armed Bandit

A multi-armed bandit [28] is a specific kind of decision problem. We have K 'arms', each with their own possibly unobservable state. At each discrete timestep, we pull exactly one of the arms, and get reward from it. The arm we pull has its state change; the other arms have their states remain as they were.

For the mathematics I can do no better than to quote from the paper [29] responsible for the key ideas of multi-armed bandits:

"There are K arms which may be pulled repeatedly in any order. Each pull takes one time unit and only one arm may be pulled at a time. A pull may result in either a success or a failure. The sequence of successes and failures which result from pulling arm i forms a Bernoulli process with an unknown success probability θ_i, $(i = 1, 2, \ldots, K)$. A successful pull on any arm at time t yields a reward $\beta^t (0 < \beta < 1)$, whilst an unsuccessful pull yields a zero reward. ... The problem is to decide which

arm to pull next at each stage so as to maximize the total expected reward from an infinite sequence of pulls."[*]

In this formulation, a classical multi-armed bandit with 0/1 rewards, each arm's internal state is given by the number of failures and successes accrued; rewards are discounted at a rate β, and are Bernouilli with a fixed probability for each arm. This is called a binary bandit, and is in many ways the simplest possible form.

This can be naturally extended by making the internal states of each arm more complicated. We denote arm i's state by x_i, and the total state of the system by $X = (x_1, \ldots, x_K)$. There are only two constraints on the evolution of the states: the states must evolve in time in a Markovian manner, and if an arm is not pulled, its state remains the same. We write $a(t) = 1, \ldots, n$ to indicate which arm we activate at time t.

Similarly, we can have more general reward distributions: with each arm i is associated a (possibly randomised) reward function $g_i(x_i)$, and at time t we get reward $g_{a(t)}(x_{a(t)}(t))$: we get no reward from non-activated arms. Replacing the Bernoulli distributions with others, possibly with more parameters is simple. Alternatively, we can make the rewards depend on the state of the arm, possibly in addition to some parameters. We would generally know the state of each arm, but initially be unsure of some of the parameters.

The control u must remain in the same simple form for this to be considered a bandit

[*]The notation has been altered slightly to fit with ours.

problem: at each stage we pull precisely one arm.

Bandit problems originated in the study of allocation problems; the prototypical such problem is that of a clinical trial [28, 30], where we can give patients either a new treatment or an old one. These correspond to having two arms, and we wish to maximise the number of patients treated successfully. This contrasts with standard trial-design procedures, which specify the allocations beforehand, and seek an unbiased estimator of the effect of the treatment.

Gittins Indices

The surprising quality of the bandit problem formulation given above is that it admits a simply-expressible optimal policy. This is an index policy: at each timestep, for each arm i we calculate an index $G(i, x_i)$ and then choose to activate the arm with the highest index, recalculating after each pull. It is easiest to give the formulation first, and then to explain what it means. For arm i, currently in state x_i, the Gittins index is:

$$G(i, x_i) = \sup_{\tau \in \mathbb{N}} \frac{\mathbb{E}[\sum_{t=0}^{\tau-1} \beta^t g_i(x_i(t))]}{\mathbb{E}[\sum_{t=0}^{\tau-1} \beta^t]} \tag{1.2.8}$$

The key part here is τ. On top, we have the expected total discounted reward gained if we play this arm for τ steps, starting in state x. On the bottom, we essentially have the expected total discounted time if we play this arm for τ steps. The Gittins index

is asking: for state i, how well can we possibly do in terms of discounted expected reward per discounted expected time? We compare the different arms by how well they can possibly do.

It can be proven [29] that if, at each step, we evaluate $G(i, x_i)$ for each arm, and then pick the arm with highest index, we get the maximum expected reward; for more detail see Gittins' paper [29].

How does this relate to our problem? The answer is, not particularly well; there is no real way to incorporate both failure and replenishment into such a model. Therefore, we need to add an additional complication, by allowing the states of inactive arms to evolve as well.

Restless Bandits

The assumption in the above section that we wish to break is that the states of non-activated arms do not evolve, so as to be able to incorporate failure with replenishment. If we allow all of the arms' states to evolve at each timestep and can pull multiple arms at the same timestep, keeping everything else the same, the resultant problem is called a restless bandit [31]. The dynamics evolve differently depending on whether or not we activate a given arm. Our control a is now a vector of zeroes and ones of length K, where $a_i = 1$ if we activate arm i, and $a_i = 0$ if we do not. There is also the constraint that $\sum_{i=1}^{K} a_i \leq M$ for some integer $M < K$, so that we can only activate M arms at each timestep.

As all of the arms are evolving even if not activated, it is reasonable for reward to be gained from all arms at each timestep, probably depending on both the control applied and the arms' states. We write the reward gained at each step as $R(x, a) = \sum_{i=1}^{K} g(x_i, a_i)$, so that rewards are still gained independently from each arm.

This is now more obviously akin to our problem as outlined in Section 1.1.1. If we let the 'arms' be the tasks, then we activate an arm by sending an asset to it. As long as the reward is gained independently from each task, we can see this as a restless bandit problem. Of course this would require a constant number of assets to become deployable per timestep, but details such as that can be folded into the implementation, and will be in Chapter 2.

Whittle Indices

In a paper [31], Peter Whittle generalised the Gittins index for restless bandits. The approach was highly innovative, and uses several tricks that we later apply in a very similar context, so it is worth going through the derivation.

First, some notation. Each arm i still has its state x_i. This evolves according to a transition-probability matrix P_{i0} if not activated and P_{i1} if activated, so that we can write that arm i always evolves by the matrix P_{ia_i}. We want to maximise the average reward $v = \sum v_i$ over an infinite horizon. Define $\phi(x_i)$ as being the transient value of starting arm i in state x_i. We can write the reward function for arm i as $g_i(x_i, a_i)$.

If we only had arm i to worry about, the Bellman equation would be:

$$v_i + \phi(x_i) = \max_{a_i=0,1} (g_i(x_i, a_i) + \sum_{x_j} P_i a_i[x_i, y]\phi(x_j)) \tag{1.2.9}$$

What about the constraint that $\sum a_i \leq M$? We can temporarily relax this constraint, and instead only expect it to hold in expectation, so that $\mathbb{E}[\sum a_i] \leq M$. In fact, the whole point of this limitation is that activating an arm is better than not doing so, so we put the constraint as $\mathbb{E}[\sum a_i] = M$: we activate exactly M arms in expectation.

In order to push us towards not activating an arm, suppose we get a charge* W_i for activating arm i. The formulation is now a Lagrangian relaxation of the original problem, with W_i serving the role of the Lagrange multiplier:

$$v_i + \phi(x_i) = \max_{a_i=0,1} (g_i(x_i, a_i) - W_i a_i + \sum_{y} P_{ia_i}[x_i, y]\phi(x_j)) \tag{1.2.10}$$

Now define the index $W_i(x_i)$ to be the value of W_i that makes us exactly indifferent in Equation (1.2.10) to the choice between $a_i = 0$ and $a_i = 1$. The Whittle index policy for this problem is to at each moment pick the M arms with the highest such indices: the arms that we are willing to pay the most to activate.

It's worth mentioning that, as opposed to the Gittins index, this definition is not a closed-form expression. Any application must therefore find one for the particular circumstances.

For this approach to yield a sensible policy, we need an additional condition called

*The original paper has a subsidy for passivity. This form fits better with our problems.

indexability [32, 33]. In plain English, it requires that increasing the charge for activating an arm should only decrease the number of states we activate the arm for.

Definition: Let $D_i(W_i)$ be the set of values of x_i for which we would not activate the arm in the optimal policy, given a charge W_i is applied. We say this arm is indexable if, as we increase W_i from $-\infty$ to $+\infty$, the space $D_i(W_i)$ decreases monotonically from X_i to $\emptyset$.

This condition is obviously reasonable, but not necessarily the case*, and has to be proven for any restless bandit formulation that we choose to use.

This set-up is useful for two reasons. First, if $M = 1$ and the non-activated arms do not evolve, then the Whittle index policy reduces to the Gittins index policy. Second, there are reasons [31, 33] to expect the Whittle index policy to be asymptotically optimal, as $n, M \to \infty$ with n/M in fixed proportion.

1.3 Related Work

Now that we have a proper handle on the background of the models we will use, we will look at some mathematical problems studied which are adjacent to ours. In this section, we describe several of them, how they differ from ours, and why and how our work is new. While Section 1.1.1 looked more at the practical problems our work was inspired by, this section is about looking at papers whose approaches or techniques

*See [31], Appendix.

are similar to ours.

MDPs are ubiquitous, and each of the subsections herein could be a book of itself; we therefore focus on literature where the problems considered are similar to ours, especially in the 'allocation' and 'failure' aspects, aiming to contextualise our work. We particularly want to look at the approaches used more than the problems studied, and hence how our work is different and new.

1.3.1 Classes of Allocation Problems

We start with a look at allocation problems. An allocation problem [34] is a very general term, but here we mean the following: we have assets, we allocate them to tasks and we get some reward based on the allocation. The assets are sometimes called resources or machines, while the tasks may be known as jobs or projects. Contrary to our problems, there is neither dynamism nor failure. There are many generic classes of allocation problems that have more than a passing resemblance to ours; we list a few:

- Factory location [35, 36] : we have a number of possible locations, and have to allocate a smaller number of factories to them, to maximise some reward, often related to the coverage of the area.
- Job scheduling [37, 38, 39]: we have a number of jobs to assign to a set of machines, and usually want to complete all tasks as fast as possible.

- Multiple knapsack [40, 41, 42]: we have a number of objects to fit into several containers, constrained by size, and want to fit in the maximum value possible.

We now go through a few specific fields in more detail, aiming to look for lessons we can learn about how we should proceed; this will help us answer the questions raised in Section 1.1.2 above.

1.3.2 Queueing Theory

If we restrict our attention to a single task, with allocations and failures, what we have is mathematically a queue. Queueing theory is a surprisingly old field, originating in the study of telephone lines in the 1920s [43], with much work still done on telecommunications [44, 45, 46, 47]. Given the variety of circumstances in which queues occur, there are thousands of published papers using or studying queueing theory. As mentioned above, we will look at only a few papers, and will primarily be interested in how they compare to what we wish to do.

Generically with queues, we have either no control or limited control of arrivals and services (the equivalent of failures), and seek either to analytically find some measure of performance such as average waiting time, or to optimise the same. The key point is that finding the relevant value is often as hard as optimising it [48]. Queueing systems also exist [49, 50, 51, 52]: a system of queues, linked in some way. Often departures from one queue form the arrivals to another; we later consider this sort of multi-phase structure (as mentioned in Section 1.1.2 above), so the concept of a

queueing system is worth bearing in mind.

We should clarify that in queueing theory, the term 'multi-phase' refer to the dynamics of a single queue, while several parallel queues are usually called a queueing system.

One paper typical of the field [53] considers a single $M/G/1$ queue: arrivals are Markovian (so form a Poisson process) and service times have a General distribution: any continuous-time strictly positive random distribution is acceptable. The '1' means that there is only one server. Unlike much of the queueing theory literature, this paper does consider the control of the system. The single server can fail, and is then repaired, and can go back to service.

The control considered is quite limited*. They set a threshold N for the size of the queue. If the queue ever empties, the server goes on holiday for some random time. When they return, they immediately take another vacation if there are fewer than N in the queue. What the authors assume is that the expected cost is convex in terms of N; this means the cost can be calculated for each N in term, and once the cost is larger for $N+1$ than for N, that value of N must be optimal.

Even though this paper does consider the control of the queue, most of the paper is concerned with the dynamics of the system. In addition, they only consider a single queue, and the control scheme is oddly limited. An older but similar paper is [55], which does not allow for random failures of the server.

To look at a recent paper about queueing systems, we will take [56]. This is a review

*But apparently quite common in the literature. See [54] for another example.

paper, and so covers typical techniques used to models queues of airplanes at airports. A key aspect is that the queues are generally non-stationary - planes do not always arrive at the same rate. This is self-evidently correct, but messes with standard techniques to an unhelpful degree.

Many airports have more than one runway, and so a single airport is often represented as a queueing system, not a single queue. Another aspect of the modelling is the creation of discrete time - the system is modelled in discrete 'slots of 15 minutes or an hour. This paper also discusses the usual modelling of arrivals as a Poisson process, or using more general distributions - this is what we talked about in our context in Section 1.1.2 above.

The key takeaway from this section is the importance of modelling: the literature tends to stick to more tractable models, even at the expense of realism, as this allows mathematical modelling to be done. We will do the same, but our problems are more general than standard queues in both the way in which we get rewards, and in the linkage of failure and replenishment.

1.3.3 Multi-Agent Models

If we only had one asset, allocating it would be fairly simple, and the dynamics of failure and replenishment would be even simpler: it is either working or it is not. The presence of multiple assets is therefore necessary for an interesting problem. What has been looked at often in the literature are multi-agent models, where several agents

(assets) work to achieve some goal. We will now review a few of the key studies here, in the context of multiple agent patrol, search and surveillance, which is particularly relevant to our naval search setting. [2] is a good overview of multiple-UAV search and surveillance problems; [57] is also worth a look more generally.

A good general look at multi-agent patrol problems is [58], whose relevance is obvious: searching and patrolling are extremely closely linked. 'Patrolling problems' is quite a general heading, and the techniques which are reviewed are just as varied. This paper lists *"heuristic agents, negotiation mechanisms, reinforcement learning techniques, techniques based on graph theory and others."* as some of the approaches applied.

Although the general model of a patrolling problem is nearly always a graph, which the patrollers must traverse, there is much variation in the objective. This could be to minimise how long it is on average between visits to each node, or to minimise the maximum time any node goes between visits, amongst others.

Another more recent paper looking at a multi-agent patrolling is [59]. This paper is particularly concerned with how to optimise patrolling when the environment is changing with time - when we have nonstationarity. They nevertheless stick to a Markovian formulation, allowing the underlying graph to have its edges evolve in a Markovian manner.

Their approach is largely heuristic, but building on work from the stationary case, and their policies are evaluated through simulation. These are two aspects common in more complicated (non-stationary, heterogeneous, non-Markovian) cases for this type

of problem. We use both aspects in Chapter 3, when we get to dependent reward structures and a non-Markovian model, because standard methods break down and no longer work.

There are many more papers we have not the space to describe; [60, 61, 62, 63] are a good selection of the papers using the standard techniques in the field, while [2] again helpfully summarises current work. Where our work differs is in the detailed consideration of failure-with-replenishment, and in the specific heuristics used.

1.3.4 Dynamics of Failure

Similar to the previous section, much work has looked at the problem of analysing a complex system, and determining the chances of failures occuring. This is especially common in telecommunications networks [43, Chapter 7], [64, 65], where what matters is not 'do the lines work?' but 'does the system work?'. These studies tend to be more concerned with predicting and diagnosing failure than attempting to optimise matters.

That being said, there is a lot of money in accurately diagnosing and minimising failure, so it is hardly surprising how much it has been studied, even for some quite esoteric yet important systems [66, 67].

A helpful review paper is [68], which looks at the optimisation of maintenance, not in the practical sense of doing individual operations well, but in the operational research sense of when we should perform maintenance acts. They begin by discussing the

evident economic importance and practical inevitability of maintenance policies, and hence the need to do it as well as possible. They mention how controlling maintenance costs is common, but ensuring that this does not come at other costs (delays, failures) is much more complex.

The main part of the paper is concerned with classiyfing the literature, noting that there is a clear divide between the academic research on the topic and the actual cases of industrial usage of such work.

A more recent paper with particular relevance is [69]. This is about recovery from failures in an airline setting, and has similar elements of dynamicism and failure to our problem. The failures here are anything that cause a flight to be delayed or cancelled, and the dynamic aspect is provided by the relentless passage of time: a flight is cancelled, but we still need to ensure that subsequent flights proceed with a minimum of disruption.

The approach here is unusual, using simulation to model the effects of possible decisions, and then picking based on what does best in the simulations. For the simulations, they have to bow to the reality that high-fidelity modelling of the whole system - all planes and airports in a given region - is simply impractical. The clever trick is in the title “Multi-fidelity Modelling’. This is of course more complicated than those three words, but the essential idea is to concentrate on the important bits.

There is a lesson here: you can’t model everything, so pick and choose the significant features. The specific features that distinguish our work are the coupling of failure

and replenishments and the focus on optimising the rewards gained, not merely the maintenance procedures.

1.3.5 Bandit Problems

As mentioned in 1.2.2, bandit problems originated in the design of medical trials [28, 29, 30], but have been used for much else [70, 71, 72, 73, 74]. While we mainly look at restless bandits, it is worth mentioning that bandit problems have often been studied in a military context; see [75, 76, 77] or [78]. We talk more about bandit problems in Section 2.1.1.

Quite a few restless bandit models are adjacent to ours [79, 80], at least for the Markovian formulations of our problems, and we use a restless bandit model for one version of our problem in the next chapter. The following paper [81] is a good recent example, and quite interesting on its own merits. This paper is concerned with the theoretical analysis, and to quote its abstract:

"we derive an analytical expression for Whittle's index for any Markovian bandit with both finite and infinite transition rates"

As we mentioned above in Section 1.2.2, there is not automatically an analytic expression for the Whittle index for a given restless bandit problem. Therefore, any application of Whittle indices, including ours, has to start by either finding or approximating the indices for the problem at hand. The derivation in this paper is therefore a significant achievement.

The authors go on to extend their work to describe the conditions for indexability; again, indexability is seemingly obvious but not in fact guaranteed. We have to prove it for our problem through somewhat lengthy algebra.

A practical application of restless bandits is given in [82]. They consider a scheduling problem with deadlines which impose a penalty if not met*. This could arise from charging a set of drones or electric cars, which need to be ready to go by a specific time. There is much randomness in the problem: *"key parameters of the problem such as job arrivals, workloads, deadlines of completion, and processing costs are stochastic"*.

The model used is a restless multi-armed bandit. They derive a closed-form expression for the Whittle indices, at least for a limited case, and prove indexability. They also use computer simulations to measure the performance of the Whittle index policy.

Another restless bandit application comes in [86], which looks at scheduling grouped maintenance interventions on a set of degrading machines or factories. The outline is the same: use a Lagrangian relaxation of the problem to find the Whittle indices, then do numerical experimentation to see what the optimality gap looks like. That is a common theme in restless bandit papers: relax to find Whittle indices, prove indexability, then use numerical simulation to establish performance. We follow the same structure in Chapter 2.

An earlier PhD thesis looking at restless bandit models in a similar area is [87].

*In general deadlines can be hard, where they are known in advance and are a constraint that must be met, or soft, where they are random and apply a penalty if missed. In queueing theory soft deadlines are often called impatience or abandonment. For an example of bandits with hard deadlines see [83], while for examples of soft deadlines see [84, 85].

This looked at the dynamic control of stochastic resource-sharing systems; this is quite similar to our problem, but they are concerned with sharing of resources not assets*, and the systems considered are quite different. To quote directly from the abstract, *"These type of problems have a stochastic nature and may be very complex to solve. We therefore focus on developing well-performing heuristics"*, which is again a common theme, that the sensible model is too complicated to solve exactly.

The author takes two approaches, using first a restless bandit model and then a fluid approximation, as we mentioned in Section 1.1.1. They mention specifically both the necessity of verifying indexability, and how the Whittle index is only guaranteed to be optimal asymptomatically. This approach differs from ours; we will focus on smaller numbers where fluid approximations are inappropriate, and take much more of an *ad hoc* approach to finding effective policies.

It is also worth a mention here that while in the next chapter we use a restless bandit model, we will then go on to consider more complicated dependent reward models, which are not amenable to the use of the same framework. We also are using a restless bandit model as an approximation of the true system, so we will have to consider how to adapt the indices to the system of interest.

*Although the two concepts are admittedly quite similar.

1.4 Next Steps

In the next chapter we will look at a simple model, that is 'independent' as described in Section 1.1.2: we can write the reward function as $R(x) = \sum_{i=1}^{n} g_i(x_i)$. We will look at this model first because it is amenable to the use of restless bandits.

In Chapter 3 we will then go on to consider several dependent models, where we will have to take a different approach.

Chapter 2: The Independent Case

This chapter was originally published as [1]; it has been lightly edited for this thesis.

2.1 Introduction

The allocation of assets in an efficient way is an omnipresent problem. It is challenging because of the typical feature that the number of available assets is limited; such limited assets usually refer to a workforce or machines available to perform a particular task.

When assets are relatively simple machines connected to an energy supply, such as computers, it is reasonable to assume that they are available for continuous operation. Allocation of such assets to tasks has been extensively studied in the performance evaluation literature employing queueing theory techniques (see e.g. [88]) and in the stochastic optimization literature employing dynamic programming techniques (see, e.g., [89, 90]).

However, considering a human workforce or cutting-edge machine technology, assets are rarely available for continuous operation. Such assets may be complex and hence failure-prone and time-consuming to repair, specialised and hence hard to quickly replace in case of an abandonment, or of limited endurance and hence regularly require recharging/refuelling. In this chapter we focus on the allocation of such assets that tend to fail and must go through a replenishment process before they are once again available for operation.

2.1.1 Motivating Examples

There is a broad range of situations in which assets are subject to failure.

Surveillance and Patrolling: Consider a number of Unmanned Aerial Vehicles (UAVs) deployed to monitor a large area. The controller may want to keep surveillance at all times, so as not to allow gaps that could be exploited by an adversary. In this kind of situation, one could easily have tens to hundreds of UAVs, with time-scales stretching from minutes to days. The UAVs have a finite endurance and so must periodically return to base for refueling and maintenance. For a review of the general UAV surveillance problem, see [2]; a similar situation arises in patrolling see [91].

Search and Rescue: Similar difficulties occur in search and rescue operations, with the added complication that the assets may often be of different types such as foot teams, search dogs and helicopters. In this case, there may be only a small number of targets to be found, and finding them quickly could be of extreme importance. One

such situation arises in a naval context, where UAVs can search for downed airmen in an ocean. See, for example [92] and [93]. Search assets cannot operate indefinitely; they need to occasionally rest (for humans or animals) or refuel (for vehicles).

Environment and Wildlife Monitoring: In environment and wildlife monitoring the controller needs to monitor an area for some time, either to get reliable readings or to ensure a sufficient sample of the flora or fauna in question. These are an example of why failure must inevitably be dealt with: it is simply impractical to survey the entire areas at once, and so some form of higher-level coordination is necessary. See, e.g., [94] and [20].

Vehicle Rental: The recent surge of sharing and short-term renting of environmentally-friendly vehicles (such as bicycles, drones, electric scooters, cars, etc.) in mobility-on-demand systems brings similar challenges. These vehicles are distributed to different locations where they wait (passively search) for customers. They frequently require repairs, cleaning and relocation. See, e.g. [95, 96].

Queueing Systems: More generically, consider a set of queues, where the controller assigns servers (assets) to a number of queues (tasks) to deal with arriving customers. The servers can only serve so many customers before needing a break, giving rise to failure and replenishment. Literature on queueing systems with server vacations and breakdowns is extensive (e.g. [97]), but their control has only been addressed in specific cases; see for example [55] or [53].

Project Management. If assets are employees or contractors, a manager needs to

assign them to projects composed of tasks. People can only work for so long before needing a break, a rest or taking leave. Such features are typically ignored in the project management literature (see [22]). There might be significantly more tasks than assets, with decisions being made on a scale of weeks or months.

2.1.2 Problem Description

In the rest of the chapter we consider a generic problem in which a number of interchangeable assets need to be allocated to a number of different tasks. Any asset is allocatable to any task and several assets can be allocated concurrently to the same task.

An asset will fail after a random amount of time performing a task. The failure may manifest itself as the asset breaking down, draining its energy supply or being damaged. The failure requires repair, which occurs via recharging, refuelling, taking a break or replacement. The repair time is also random. During this repair time the asset cannot be allocated to any task. Once an asset is repaired, it is kept in reserve ready for allocation to any task, either immediately or later. Assets cannot be freely moved from one task to another, only allocated from the reserve to any of the tasks.

Every task yields a reward at a rate depending on the number of deployed assets, which is assumed non-decreasing and concave. The objective is to find a way of allocating assets to tasks that will maximise the long-term time-average sum of per-task reward rates.

It would be possible to formulate this problem without a reserve, and require that upon being repaired, an asset must be allocated to one of the k tasks. While this would yield a smaller state-space, for many reward functions, particularly those which are strongly concave, the optimal policy does keep a number of assets in the reserve. In Section 2.3 and Section 2.4 we develop alternative policies that do not reserve. In Section 2.5 we perform a numerical comparison of the optimal policy versus these alternative policies and find that while in most scenarios the reserve option provides little benefit, in some cases it is vital.

2.1.3 Contributions and Chapter Structure

In Section 2.2 we formulate this problem as a closed-system continuous-time Markov decision problem (MDP) with impulsive controls. To overcome the theoretical issues with impulsive controls, we reformulate it equivalently by replacing the impulsive controls with actions that do not immediately lead to system transitions, and further uniformise and discretise the problem. The resulting formulation allows for numerical solution by dynamic programming of small problem instances. It also allows for defining a greedy policy which myopically maximises the sum of reward rates.

Relevant to this formulation is research on optimisation of closed systems with multiple tasks. Many papers have used the framework of queueing networks (see, e.g., [89]), but these allow or even require assets to move between tasks and only one asset is allowed to be active at a task: the other assets wait in a queue). In terms of

optimisation, the focus is mainly on queue scheduling disciplines, which do not apply in our model, and on service rate control, which is somewhat related to our model as the number of allocated assets to a task affects both the failure rate and the reward.

For any reasonably-sized problem, standard dynamic programming techniques are computationally infeasible. In Section 2.3 we therefore develop an alternative formulation of the problem as an open-system approximation to the original closed-system problem. The open system imagines that assets depart from the system when they fail and new assets arrive independently to the system, as if from the repair process.

This model is related to the standard problem of routing or dispatching to parallel queues (see, e.g. [98]), in which assets correspond to jobs and tasks to queues with servers, with the difference that each task is shared by all the allocated assets rather than following a queueing discipline. We extend the restless bandits approach of [31] to derive its Lagrangian relaxation and decomposition, which leads to a parametric single-task problem that can be seen as a problem of admission control to a multi-server queueing system; Section 2.3.2 is then devoted to the derivation of the Whittle index. A similar model and solution approach appeared in [99], with the difference that the reward from the task is constant in the number of assets deployed.

In Section 2.4 we describe two ways of adapting the Whittle index derived from the open-system model to the original closed-system model, a naïve one and a cleverly modified one. To the best of our knowledge these are both novel. In order to prepare ground for numerical evaluation of these index rules, we describe the uniformly random

policy and the greedy policy, and explain how the optimal policy can be found when it is practical to do so.

Section 2.5 then describes comprehensive numerical experiments of performance evaluation of these policies in the original closed-system model, which indicates that the cleverly modified Whittle index rule is nearly optimal, being within 1.6% (0.4%, 0.0%) of the optimal reward rate 75% (50%, 25%) of the time, and significantly superior to uniformly random allocation which is within 22.0% (16.2%, 10.7%) of the optimal reward rate (see Table 3.5.1). Our numerical results also suggest that it is crucial for the Whittle index to be modified when adapting it from the open-system to the closed-system model, since the naïve Whittle index rule is not superior to the greedy policy which is obtained by myopically optimising the original closed-system model.

Finally, Section 2.6 presents limitations and concludes with a discussion of possible extensions of this work. A single overly long proof is deferred to Appendix A, while pseudocode for the policy improvement algorithm to optimally solve the MDP is given in Appendix B.1.

2.2 The Problem Formulation

In this section we formalise the problem outlined in the previous section as a continuous-time Markov decision process (MDP) (see [24], chapter 11); that is, for the dynamics we focus on the Markovian case. There are N interchangeable assets to allocate to K

different tasks.

2.2.1 Markov Decision Process Model

A continuous-time MDP operating over an infinite time-horizon $t \geq 0$ is defined by its states, actions, transition rates, and reward rates.

States: We denote the current number of assets at task $k = 1, \ldots, K$ by x_k. For notational convenience, we write x_{K+1} for the number of assets under repair, and x_{K+2} for the number of assets currently repaired but not yet allocated – the number currently reserved. The reserve does in fact sometimes prove necessary, especially if the reward functions are very concave. Assets are discrete so that $x_k \in \mathbb{N}_0$ – the non-negative integers – for all $k = 1, \ldots, K+2$, and the total number of assets is constant so that $\sum_{k=1}^{K+2} x_k = N$. We denote the current overall state by $\boldsymbol{x} = (x_1, x_2, ..., x_{K+2})$, and the space of possible states by $\mathcal{X}$.

Actions: After it reaches the reserve task, any asset can then immediately be allocated from the reserve to any of the tasks, or be held back for later allocation. The actions $\boldsymbol{a}$ are the allocations of assets from reserves to tasks $k = 1, \ldots, K$, which move the state instantaneously from $\boldsymbol{x}$ to $\boldsymbol{x} + \boldsymbol{a}$. We write $\mathcal{A}(\boldsymbol{x})$ for the space of currently feasible actions when in state $\boldsymbol{x}$; it is defined by the following four constraints:

1. $a_k \geq 0$, $k = 1, \ldots, K+1$; previously allocated assets or assets that are being repaired may not be reallocated;

2. $-x_{K+2} \leq a_{K+2} \leq 0$; assets from the reserve can be allocated up to the number currently there;

3. $\sum_{k=1}^{K+2} a_k = 0$; the total number of assets is constant;

4. $a_k \in \mathbb{Z}\ \forall k$; assets are discrete.

We may take the action $\mathbf{0}$ even if we have assets in the reserve: we do not have to allocate all the immediately available assets. Lastly, in practice we would never want to allocate assets directly to the repair process,so that $a_{K+1} = 0$ for any sensible policy, but we still allow the possibility.

We specify the actions to be chosen for each state by a policy u. A policy u is a map, possibly non-deterministic*, from states $\boldsymbol{x} \in \mathcal{X}$ to actions $\boldsymbol{a} \in \mathcal{A}$. The family of admissible policies $\mathcal{U}$ includes all non-anticipating policies prescribing actions satisfying $\boldsymbol{a} \in \mathcal{A}(\boldsymbol{x})$. We will in particular restrict ourselves to deterministic stationary policies, which always take the same action when in the same state. This can be done without loss of generality for MDPs [24].

Transition rates: Apart from those caused by our actions, there are two types of transition: due to a failure and due to a repair. Neither the failure transition nor the repair transition depends on the current action. Failures happen on a per-asset basis, depending only on the task the asset is allocated to; we denote the failure rate for an asset allocated to task k by μ_k, with the natural constraint that all $\mu_k > 0$. Any asset that fails immediately joins the repair process, i.e. a failure in task k causes a

*But in this chapter always deterministic.

transition from $\boldsymbol{x}$ to $\boldsymbol{x} - \mathbf{e}_k + \mathbf{e}_{K+1}$. Here $\mathbf{e}_k$ is the k^{th} standard $(K+2)$-dimensional unit basis vector.

The repair process produces ready-to-allocate assets one at a time at some non-decreasing state-dependent rate $\lambda(x_{K+1})$. The function $\lambda(x)$ is a function defined on $\mathbb{N}_0$, satisfying the following two constraints: $\lambda(0) = 0$ so no assets means no repairs, and $\lambda(x)$ is non-decreasing so more assets being repaired makes repairs more likely to happen. After an asset is repaired, it transitions to the reserve status: repairs, which can happen as long as $x_{K+1} \geq 1$, cause a transition from $\boldsymbol{x}$ to $\boldsymbol{x} - \mathbf{e}_{K+1} + \mathbf{e}_{K+2}$.

The assets' time to failure and the time until the next asset is repaired both follow exponential distributions. The times-to-failure are independent for each asset and independent of anything else, such as the overall state of the system; the repair process is independent of anything else as well.

Reward rates: Since the actions move the state instantaneously, for our purposes it is sufficient to consider reward rates that are independent of actions. We write that in state $\boldsymbol{x}$, the controller gains reward at a rate $R(\boldsymbol{x})$. We impose that the rewards are gained independently from each task; that is, the reward-rate function is of the following form:

$$R(\boldsymbol{x}) = \sum_{k=1}^{K} g_k(x_k) \tag{2.2.1}$$

The functions g_k represent the reward obtained from allocating assets to task $k =$

$1, \ldots, K$. We impose three conditions of the g_k, which we expect to hold in practice:

1. $g_k(x)$ is non-decreasing; more assets allocated to a task produces no less reward;
2. $g_k(x)$ is concave in x; there are diminishing marginal rewards;
3. $g_k(0) = 0$; no assets allocated gives no reward.

There is no reward gained for assets under repair or in reserve.

Objective: The MDP is characterised by the state process $\boldsymbol{X}(\cdot)$ and the action process $\boldsymbol{A}(\cdot)$, induced by the transition rates and by the chosen policy u. We consider the long-term time-average objective for the MDP over an infinite horizon. We define V as the optimal long-term time-average reward rate, so that:

$$V = \max_{u \in \mathcal{U}} \liminf_{H \to \infty} \frac{1}{H} \mathbb{E}^u \left[\int_0^H R(\boldsymbol{x}(t)) dt \right] \tag{2.2.2}$$

This expression is well defined because the state-space is finite and the reward rates are non-negative and bounded above, since there is a finite number of assets.

2.2.2 Solution by Dynamic Programming

The type of actions defined above are known as impulsive controls for continuous-time MDPs. The main difficulty with dealing with impulsive control models is that the

process can take several different values at the same time moment, which makes the classical theory of MDPs inapplicable.

The structure of our MDP model allows for a reformulation in which impulsive actions are replaced by actions that do not immediately lead to system transitions, by making both the transition rates and reward rates dependent on the action, as in [27]. This reformulation results in a merging of the state transition caused by the impulsive action with the next state transition due to failure or repair. Rather than viewing the system as transitioning instantaneously from $\boldsymbol{x}$ to $\boldsymbol{x}+\boldsymbol{a}$, we frame the system as remaining in state $\boldsymbol{x}$ with rewards and transitions dictated by $\boldsymbol{x}+\boldsymbol{a}$.

In particular, a failure of an asset allocated to task k while $\boldsymbol{a}$ prevails occurs at rate $(x_k + a_k)\mu_k$ and causes a transition from $\boldsymbol{x}$ to $\boldsymbol{x}+\boldsymbol{a}-\mathbf{e}_k+\mathbf{e}_{K+1}$. Similarly while $\boldsymbol{a}$ prevails, a repair occurs at rate $\lambda(x_{K+1}+a_{K+1})$ and causes a transition from $\boldsymbol{x}$ to $\boldsymbol{x}+\boldsymbol{a}-\mathbf{e}_{K+1}+\mathbf{e}_{K+2}$. Finally, while $\boldsymbol{a}$ prevails in state $\boldsymbol{x}$ the controller gains reward at a rate $R(\boldsymbol{x}+\boldsymbol{a})$.

As we have instant controls, applying two actions a_1 and then a_2 is equivalent to applying $a_1 + a_2$.

We can uniformise the problem, transforming the reformulated continuous-time model into an equivalent discrete-time model. Define $B = \lambda(N) + N \max \mu_k$, as an upper bound on the transition rate out of all states. Consider a related Markov decision process that transitions out of every state with rate B. For this related process, with probability $(x_k + a_k)\mu_k/B$ an asset in task k fails and with probability $\lambda(x_{K+1} +$

$a_{K+1})/B$ an asset is repaired. With the remaining probability $[B - \lambda(x_{K+1} + a_{K+1}) - \sum_{k=1}^{K}(x_k + a_k)\mu_k]/B$ the dummy transition moves the system to state $\boldsymbol{x} + \boldsymbol{a}$. During uniformisation the dummy transition usually has the system return to its current state; because of our impulsive action reformulation described in the previous paragraph, the dummy transition here goes to $\boldsymbol{x} + \boldsymbol{a}$. For more details on uniformisation, see [24, section 11.5] .

This uniformisation results in a discrete-time MDP. We can present the optimality equation for the discrete-time MDP with a long-term time-average objective, utilising standard MDP machinery; see [24, Chapters 8 and 11] for more details. We define $\phi(\boldsymbol{x})$ as the relative value of a state $\boldsymbol{x}$. This relative value is also referred to as the bias (see [24, Chapter 8]), and is a measure of how much the controller prefers starting the system in one state compared to some reference state.

We obtain the following dynamic programming optimality equation, which is satisfied by the optimal value V for our problem:

$$
\begin{aligned}
V + \phi(\boldsymbol{x}) = \max_{\boldsymbol{a} \in \mathcal{A}(\boldsymbol{x})} \frac{1}{B} \Bigg(& R(\boldsymbol{x} + \boldsymbol{a}) \\
& + \lambda(x_{K+1} + a_{K+1}) \phi(\boldsymbol{x} + \boldsymbol{a} - \mathbf{e}_{K+1} + \mathbf{e}_{K+2}) \\
& + \sum_{k=1}^{K} (x_k + a_k) \mu_k \phi(\boldsymbol{x} + \boldsymbol{a} - \mathbf{e}_k + \mathbf{e}_{K+1}) \\
& + \left(B - \lambda(x_{K+1} + a_{K+1}) - \sum_{k=1}^{K} (x_k + a_k) \mu_k \right) \phi(\boldsymbol{x} + \boldsymbol{a}) \Bigg) \quad \forall \boldsymbol{x} \in \mathcal{X}
\end{aligned}
\tag{2.2.3}
$$

This is a set of $|\mathcal{X}|$ equations for $|\mathcal{X}|+1$ unknowns - the $\phi(x)$ and V. We hence choose some reference state $\mathbf{x}_0$ (in practice $(0, 0, \ldots, N)$) and set $\phi(\mathbf{x}_0) = 0$. We can then solve the set of equations to find the optimal policy, and the optimal value V.

For definiteness, we define $\phi(\boldsymbol{x}) = 0$ for all $\boldsymbol{x}$ such that $x_k < 0$ for any $k = 1, \ldots, K+2$: for all ineligible x the terms disappear.

On the LHS of Equation (2.2.3), we have the optimal value plus the relative value of the current state. On the RHS, we choose the control $\boldsymbol{a}$ to maximise the sum of four terms. The first term is the immediate expected reward received in state $\boldsymbol{x}$ while $\boldsymbol{a}$ prevails. Note that $R(\boldsymbol{x}+\boldsymbol{a})$ is a rate, and hence $R(\boldsymbol{x}+\boldsymbol{a})/B$ is the expected reward accumulated before the next transition.

The second term captures the expected future reward obtained when a repair transition occurs next. The third term represents the expected reward from all failure

transitions. The fourth term is the expected reward from the dummy transition where the system moves to state $\boldsymbol{x} + \boldsymbol{a}$.

In theory, we can compute the optimal V from Equation (2.2.3) using standard MDP solution techniques such as policy iteration, value iteration, or linear programming (see [24] chapter 8). Unfortunately, in practice such exact methods quickly become intractable even for relatively small problems.

The state space for our problem has size $|\mathcal{X}| = \binom{N+K+1}{K+1}$ because the controller distributes N identical assets among $K+2$ non-identical tasks. For instance, if $N = 15$ and $K = 10$, then $|\mathcal{X}|$ is more than 7.7 million. Given this, computing the optimal solution will only be feasible for small problems, as standard policy iteration requires us to form a matrix of size $|\mathcal{X}| + 1$ by $|\mathcal{X}| + 1$. We now turn to heuristic methods, in particular one that will allow us to consider the tasks separately.

2.3 A Restless Bandit Approximation

Restless bandits [31] provide a framework applicable to our problem. A generalisation of the well-known multi-armed bandit problem [29], a restless bandit is a specific type of MDP.

In a restless bandit problem there are K bandits, which are independent Markov decision processes with binary action space: the controller can choose to activate the bandit, or not to. Each bandit has its own state x_k and generates rewards at a rate

depending only on its own state. The control is quite simple: at each decision point, the controller can activate up to $M < K$ of the bandits; the controller gains reward according to both the state of all the bandits and the actions taken. Taking the active action is usually better in some way, either providing more immediate reward, or charging up the bandit.

The restlessness is as follows: even the bandits not activated still evolve stochastically, usually decaying in some way. This contrasts with a classic bandit problem of the Gittins type, where the controller can only activate one bandit at a time, and the non-activated bandits remain in the same states.

2.3.1 The Approximating System

We now formulate an approximating system, which modifies the original problem in three respects:

1. We remove the repair process, and have assets arriving for allocation to tasks at a constant rate Λ independently of the system's state: assets arrive according to a Poisson process with rate Λ.

2. Failing assets depart the system entirely, as opposed to going to a repair buffer.

3. We remove the reserve, so arriving assets are immediately allocated.

The diagrams in Figure 2.3.1 and Figure 2.3.2 illustrate the difference between the original system and the approximating system.

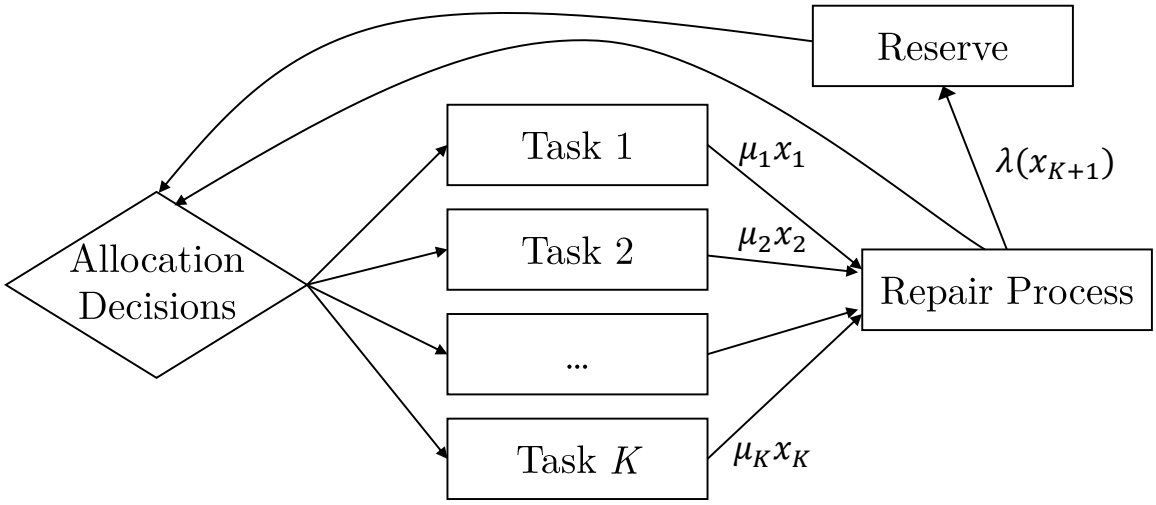


Figure 2.3.1: A diagram of the problem. Assets fail, are repaired, and are then allocated to tasks, possibly via a sojourn in the reserve.

The approximating system is a continuous-time MDP with the elements as follows. We write x_k for the number of assets currently at task $k = 1, \ldots, K$. The number of assets is unbounded and the state space is $\mathbb{N}_{\geq 0}^K$ – a vector of K non-negative integers. Each arriving asset is immediately allocated to one of K tasks. Assets at task k fail at rate μ_k independently of each other, and of everything else in the system. The aggregate asset failure rate in system state $\boldsymbol{x} = (x_1, x_2, \ldots, x_K)$ is given by $\sum_{k=1}^K \mu_k x_k$. Rewards are earned in state $\boldsymbol{x}$ at a rate $R(\boldsymbol{x}) = \sum_{k=1}^K g_k(x_k)$.

We now write $u : \mathbb{N}^K \to \{1, 2, \ldots, K\}$ for a stationary allocation policy; when the approximating system is in state $\boldsymbol{x}$, an arriving asset is assigned to task $u(\boldsymbol{x})$. We define $\mathcal{U}$ as the set of all such policies. We will write the optimum allocation problem for the approximating system as:

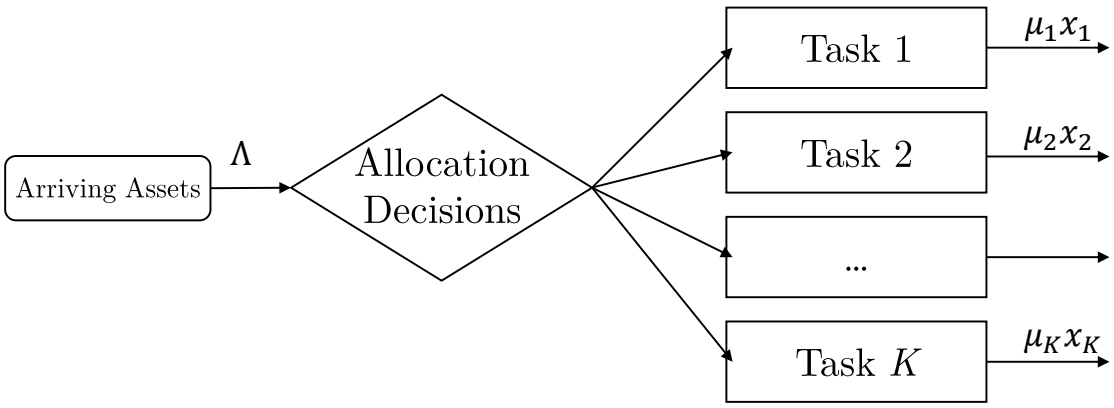


Figure 2.3.2: A diagram of the restless bandit approximation. Assets arrive at a constant rate, and there is no reserve or repair process.

$$G^{opt} = \max_{u \in \mathcal{U}} \sum_{k=1}^{K} V_k^u \tag{2.3.1}$$

Here we write V_k^u for the long-term time-average reward rate earned by task k under allocation policy $u \in \mathcal{U}$, i.e.:

$$V_k^u := \lim_{H \to \infty} \frac{1}{H} \mathbb{E}^u \left[\int_0^H g_k(X_k(t)) dt \right]. \tag{2.3.2}$$

Even this simplified version of our original problem lies beyond the scope of analysis via conventional dynamic programming. However, we can make progress by using ideas which have their origins in the description of a class of restless bandit problems in [31]. We create a Lagrangian relaxation of the above allocation problem as follows: first extend the class of feasible policies from $\mathcal{U}$ to a class $\mathcal{U}' \supseteq \mathcal{U}$.

Any $u \in \mathcal{U}'$ is given by a map $u : \mathbb{N}^K \to 2^{\{1,2,\ldots,K\}}$. Under $u \in \mathcal{U}'$ we allow each incoming asset to be allocated to all tasks in any subset (including the empty set) of the task set $\{1, 2, \ldots, K\}$, with the total number of assets assigned to each task in the nominated subset incrementing by one. Consequently, the arrival of one asset to the system can result in the allocation of up to K assets (one asset per task) for a policy $u \in \mathcal{U}'$.

This relaxation follows the form of that used in problems of routing arrivals to parallel queues, as in [73].

We write $\Lambda_k(u)$ for the long-term time-average allocation rate of assets to task k under $u \in \mathcal{U}'$. We develop a relaxation of the above allocation problem in Equation (2.3.1) as follows. Define:

$$G^{opt'} = \max_{u \in \mathcal{U}'(\Lambda)} \sum_{k=1}^{K} V_k^u \tag{2.3.3}$$

The maximum is taken over the policy set $\mathcal{U}'(\Lambda)$ given by:

$$\mathcal{U}'(\Lambda) = \left\{ u \in \mathcal{U}'; \sum_{k=1}^{K} \Lambda_k(u) \leq \Lambda \right\} \tag{2.3.4}$$

Problem Equation (2.3.3) allows us to use the extended policy class $\mathcal{U}'$, but our total asset allocation rate is constrained by the arrival rate Λ of assets.

We now relax the problem in Equation (2.3.1) further by dropping the constraint

in Equation (2.3.4) and instead incorporating it into the objective in a Lagrangian fashion to obtain

$$G^{opt}(W) = \max_{u \in \mathcal{U}'} \left\{ \sum_{k=1}^{K} (V_k^u - W\Lambda_k(u)) \right\} + W\Lambda \tag{2.3.5}$$

The multiplier W has an interpretation as a charge levied whenever an asset is allocated to a task. It is plain that

$$G^{opt}(W) \geq G^{opt'} \geq G^{opt}, \quad W \geq 0, \tag{2.3.6}$$

Further, it can be shown by a standard argument that:

$$\min_{W \geq 0} G^{opt}(W) = G^{opt'} \tag{2.3.7}$$

Further inspection of the problem in Equation (2.3.5) makes it plain that due to independence of the tasks, the problem can be decomposed into K independent asset-allocation problems, given the Lagrange multiplier W. We write:

$$G^{opt}(W) = \sum_{k=1}^{K} G_k(W) + W\Lambda, \tag{2.3.8}$$

where:

$$G_k(W) = \max_{u_k \in \{0,1\}^{\mathbb{N}}} (V_k^{u_k} - W\Lambda_k(u_k)), 1 \leq k \leq K. \tag{2.3.9}$$

The $\{0,1\}^{\mathbb{N}}$ in Equation (2.3.9) is a way to represent a policy u_k as an infinite dimensional binary vector where the policy assigns an incoming asset to task k in state i if index i of the binary vector is a 1, otherwise the policy declines the incoming asset if index i is 0.

The problem in Equation (2.3.9) concerns task k only. It seeks an optimal allocation policy when assets arrive at task k according to a Poisson process of rate Λ and are either allocated to the task (action 1), or not (action 0) dependent upon the task's current state. Reward rates for task k are as in the original problem in Equation (2.3.1), but now rewards are earned net of charges levied at a rate of W per allocation.

We can now focus attention on this parametric single-task problem, and until further notice shall drop the task identifier k from the notation. We thus write the single task problem as:

$$G(W) = \max_{u \in \{0,1\}^{\mathbb{N}}} (V^u - W\Lambda^u) \tag{2.3.10}$$

See Figure 2.3.3 for a diagram of the single-task problem. We will say that the single task problem is in a state x if there are x assets currently at the task.

We write $u(W)$ for an optimal allocation policy for the sub-problem in Equation (2.3.10). We seek optimal policies of simple structure for the Lagrangian relaxation in Equation (2.3.5) by looking for simplicity of structure in $u(W)$. The following notion [31] is key:

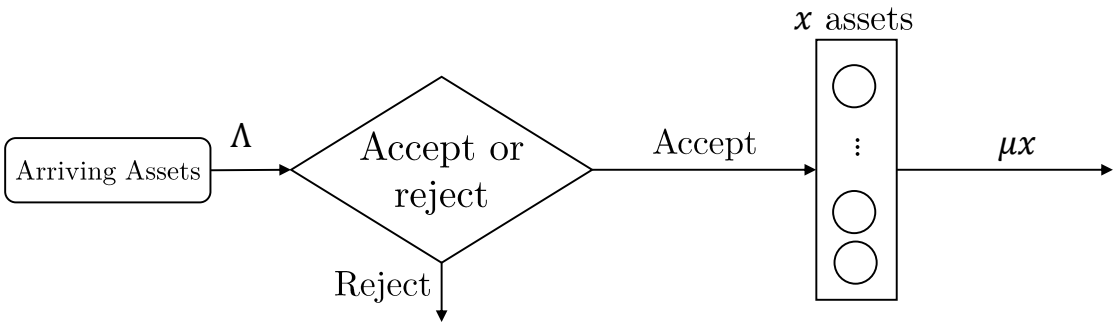


Figure 2.3.3: A diagram of a single task of the restless bandit approximation

Definition: The approximating system is indexable if there exist optimal policies $u(W)$ for all parametric single-task problems for which the activation sets:

$$A(W) = \{x | x \in \mathbb{N} \text{ and } u(W, x) = 1\} \tag{2.3.11}$$

are decreasing in W.

We prove in the following subsection that the approximating system is indexable and derive the corresponding Whittle index.

2.3.2 Indexability

Indexability is the natural requirement that, for each task, as the allocation charge W increases, the set of states in which it is optimal to allocate an incoming asset to the task decreases. Since the following result has a proof along the lines of a corresponding result in [73] for a different system, we shall provide a sketch only.

Proposition: The approximating system is indexable.

Proof: First, it is trivial to show that there exists optimal policies $u(W)$ of threshold type, namely whose corresponding allocation sets take the form:

$$A(W) = \{0, 1, \ldots, T(W) - 1\} \tag{2.3.12}$$

for some $T(W) \in \mathbb{N}$. Hence $T(W)$ is the minimal state for which the optimal policy dictates that an incoming asset should be rejected. Define $\pi_{T(W)}(x)$ as the stationary probability of being in state x, given a policy with threshold $T(W)$. For such a policy we have a long-term time-average rate of allocation of incoming assets equal to:

$$\Lambda(u(W)) = \Lambda\left\{1 - \pi_{T(W)}(T(W))\right\} \tag{2.3.13}$$

Here $\pi_{T(W)}(T(W))$ is the stationary probability that the task operating under policy $u(W)$ has its maximal number of assets present, namely $T(W)$. The dynamics of the task under this policy are a $M/M/\infty$ queue, truncated at $T(W)$, and so:

$$\pi_T(T) = \frac{\frac{1}{T!}(\frac{\Lambda}{\mu})^T}{\sum_{x=0}^{T} \frac{1}{x!}(\frac{\Lambda}{\mu})^x} \tag{2.3.14}$$

which is decreasing in T. We now remark that $G(W)$, being the upper envelope of a set of linear functions, must be convex in W. It then follows that $\Lambda(u(W))$ must be decreasing in W and hence from Equation (2.3.13), $\pi_{T(W)}(T(W))$ must be increasing in W. It then follows that $T(W)$ and $A(W)$ must be decreasing in W. This establishes

indexability, and so completes the proof.

Remark: Note that the reward function g plays no role in the proof of the above proposition. Hence the approximating system is indexable whatever the functional form the rewards take. However, as we develop stronger notions of indexability, the increasing concave nature of g will come into play. We now proceed to define the indices which the indexability of the approximating system make possible.

Definition: An index function $W : \mathbb{N} \to \mathbb{R}^+$ for a task with associated optimal acceptance sets

$\{A(W) = \{0, 1, \ldots, T(W) - 1\}, W \in \mathbb{R}^+\}$ is defined by

$$W(x) = \sup \left\{ W | W \in \mathbb{R}^+ \text{ and } x \in A(W) \right\} \tag{2.3.15}$$

Definition: The approximating system is strictly indexable if, for all tasks, the mapping $T : \mathbb{R}^+ \to \mathbb{N}$ which takes W to $T(W)$ is onto $\mathbb{N}$.

Remark: We already know from the above proposition that the mapping T is non-increasing.

Before proceeding further, we extend the notation in Equation (2.3.14) by noting that the stationary distribution for the number of assets at a task operating under acceptance set $\{0, 1, \ldots, T - 1\}$ is given by

$$\pi_T(x) = \frac{\frac{1}{x!}(\frac{\Lambda}{\mu})^x}{\sum_{y=0}^{T} \frac{1}{y!}(\frac{\Lambda}{\mu})^y}, 0 \leq x \leq T \tag{2.3.16}$$

We now introduce the key quantities $w(x), x \in \mathbb{N}$, given by

$$w(x) = \frac{\sum_{y=0}^{x+1} g(y)\pi_{x+1}(y) - \sum_{y=0}^{x} g(y)\pi_x(y)}{\Lambda(\pi_x(x) - \pi_{x+1}(x+1))} \tag{2.3.17}$$

The quantity $w(x)$ has a natural interpretation as follows [100]: imagine that the decision maker facing the single task problem in Equation (2.3.10) is committed to accepting newly arriving assets provided there are no more than $x-1$ of them currently present at the task. The question arises of the effect of expanding the acceptance set to include x as well.

The numerator in Equation (2.3.17) is the increase in reward rate thus achieved while the denominator is the increase in acceptance rate. It is the case that when this marginal quantity is less than W (the charge per unit asset accepted) then the expansion of the acceptance rate proposed leads to a decrease in the overall reward rate in Equation (2.3.10).

Should W be less than $w(x)$, then the reverse conclusion holds and the expansion proposed improves the overall reward rate. From these considerations it follows that, under suitable conditions on the reward function g, $w(x)$ coincides with the index $W(x)$ defined above. We now outline a proof of this key result.

Theorem: If the reward function g is increasing and strictly concave, then the approximating system is strictly indexable with index given by $W(x) = w(x), x \in \mathbb{N}$.

Proof: We introduce the quantities:

$$P_x = \sum_{y=0}^{x} \frac{\left(\frac{\Lambda}{\mu}\right)^y}{y!}, \qquad x \in \mathbb{N} \tag{2.3.18}$$

In Appendix A, we prove the identity:

$$w(x) = \frac{\sum_{y=0}^{x} P_y(g(y+1) - g(y))}{\mu \sum_{y=0}^{x} P_y} \tag{2.3.19}$$

from which it follows that $w(x)$ is strictly decreasing in x when g is increasing and strictly concave. We also note that $w(0) = \frac{g(1)}{\mu}$ and because g is concave, each asset considered separately can only give a reward of at most $\frac{g(1)}{\mu}$. We can therefore write:

$$0 \leq \lim_{x\to\infty} (g(x+1) - g(x)) \leq \lim_{x\to\infty} w(x) \leq \frac{g(1)}{\mu} \tag{2.3.20}$$

We now write $G_x(W)$ for the reward rate achieved in Equation (2.3.10) when the acceptance set is $\{0, 1, \ldots, x\}$. We have

$$G_x(W) = \sum_{y=0}^{x+1} g(y)\pi(y|x+1) - W\Lambda(1 - \pi(x+1|x+1)) \tag{2.3.21}$$

and we then obtain $W = w(x)$ as the unique $W-$solution to the equation:

$$G_x(W) = G_{x-1}(W), \quad x \in \mathbb{N} \tag{2.3.22}$$

with the natural interpretation for the case $x = 0$. Further, each $G_x(W)$ is linear in W with a gradient which is negative, and which decreases as x increases. From these facts it follows that for all choices of $x \in \mathbb{N}$:

$$G_x(W) < G_y(W), \quad y \leq x - 1, W \geq w(x) \tag{2.3.23}$$

and

$$G_{x+1}(W) < G_y(W), \quad y \leq x, W \leq w(x + 1) \tag{2.3.24}$$

From these inequalities and the fact that $w(x)$ is strictly decreasing in x, we can deduce that for any $x \in \mathbb{N}$:

$$G_x(W) > \max_{y \neq x} G_y(W), \quad w(x + 1) \leq W \leq w(x), \tag{2.3.25}$$

Hence:

$$A(W) = \{0, 1, \ldots, x\}, \quad w(x+1) \leq W \leq w(x) \tag{2.3.26}$$

with

$$A(w(x)) \in [\{0, 1, \ldots, x-1\}, \{0, 1, \ldots, x\}] \tag{2.3.27}$$

The corresponding rejection thresholds are given by:

$$T(W) = x + 1, \quad w(x+1) \leq W \leq w(x) \tag{2.3.28}$$

and

$$T(w(x)) \in \{x, x+1\}. \tag{2.3.29}$$

It now follows from the above that the map T is onto $\mathbb{N}$, that the approximating system is strictly indexable, and that the index is given by:

$$W(x) = \sup\left\{W | W \in \mathbb{R}^+ \text{ and } x \in A(W)\right\} = w(x) \tag{2.3.30}$$

as required. This concludes the proof.

To continue the discussion, we now restore the task identifier k. From the above discussion, under the assumption that all reward functions g_k are increasing and strictly concave, the approximating system is strictly indexable, and associated with each task k is an index function $W_k : \mathbb{N} \to \mathbb{R}^+$ given by a suitable form of the quantity in Equation (2.3.17).

It is an immediate consequence of the above discussion that an optimal policy for the Lagrangian relaxation in Equation (2.3.5) may be expressed as follows: when the approximating system is in state $\boldsymbol{x}$, an arriving asset should be allocated to those tasks for which $W_k(x_k) > W$. Denote this policy by $Index(W)$. Plainly the asset allocation rate associated with $Index(W)$ is decreasing in W. It is straightforward to show that the relaxed problem in Equation (2.3.3) is solved by $Index\left(W_\Lambda\right)$, where W_Λ is such that:

$$\sum_{k=1}^{K} \Lambda_k(Index(W_\Lambda)) = \Lambda \tag{2.3.31}$$

To achieve equality in Equation (2.3.31) it may be necessary to introduce randomisation into the operation of $Index(W_\Lambda)$ when a task-state pair for which $W_k\left(x_k\right) = W_\Lambda$ is encountered. In light of these optimal policies of index structure, we follow [31] in proposing a solution for Equation (2.3.1) in the form of an index rule for the approximating system as follows: when the approximating system is in state $\boldsymbol{x}$ an arriving asset should be allocated to any task $l \in \arg\max_k W_k\left(x_k\right)$.

2.4 Policies

To evaluate how well the Whittle index rule works on example problems, we will compare it against the optimal policy, when possible. However, as discussed earlier, it is only possible to compute the optimal value for smaller problems. Therefore, we also compare the Whittle index rule to two other policies which could easily be applied to larger problems. We now expound briefly on the policies, including how to actually apply the Whittle index rule to the original closed-system problem.

2.4.1 The Whittle Index Rule

Although we provide a full derivation and explanation of the Whittle index above, we touched only briefly on how to use this to make decisions for a given instance of the original closed-system problem. Recall from Equation (2.3.17) that the formula for the Whittle index for task k which has x_k assets is*:

$$w_k(x_k) = \frac{\sum_{x=0}^{x_k+1} \pi_{x_k+1}(x) g_k(x) - \sum_{x=0}^{x_k} \pi_{x_k}(x) g(x)}{\pi_{x_k}(x_k) - \pi_{x_k+1}(x_k+1)} \tag{2.4.1}$$

Defining $\rho_k \equiv \frac{\Lambda}{\mu_k}$ for the open-system, we rewrite the stationary distribution from Equation (2.3.16)

$$\pi_{x_k}(x) = \frac{\rho_k^x / x!}{\sum_{y=0}^{x_k} \rho_k^y / y!} \tag{2.4.2}$$

*Dropping the initial constant $1/\Lambda$ because it is the same $\forall\, k$.

The key input to compute the index $w_k(x_k)$ is the ratio ρ_k, which appears in Equation (2.4.1) via the stationary distribution in Equation (2.4.2). This ratio is not well defined in the original closed-system problem because Λ is not a parameter in that system. Consequently we need to define ρ_k more concretely for the original system. We examine two different Whittle index variants corresponding to different values for ρ_k; we denote one the *naïve* index and the other the *clever* index.

The naïve index sets $\rho_k = \frac{\lambda(x_{K+1}+1)}{\mu_k}$. This definition directly mimics the open-system definition of ρ_k by replacing the arrival rate Λ with the repair rate $\lambda(x_{K+1}+1)$, which is reasonable because assets arrive for allocation in the original system following a repair. We use $x_{K+1}+1$ instead of x_{K+1} because repaired assets move immediately to the reserve, and hence x_{K+1} could be zero when the allocation decision is made.

The clever index sets

$$\rho_k = \frac{\lambda(x_{K+1}+1)}{\lambda(x_{K+1}+1)+\mu_k}. \tag{2.4.3}$$

We choose this value of ρ_k for two reasons. First this value of ρ_k produces the optimal solution for the original closed-system in the case with $N = 1$ asset. As the rewards are unchanging, the optimal policy is to always allocate the asset to the same task. If the controller always allocates to task k, the asset is at that task for a fraction of the time $\lambda(1)/(\mu_k + \lambda(1))$. The controller gets a reward per unit time of

$$g_k(1)\frac{\lambda(1)}{\lambda(1)+\mu_k}, \tag{2.4.4}$$

and the optimal policy allocates to the task which makes the expression in Equation (2.4.4) the largest. In this case, the Whittle index for task k for the state $x_k = 0$ can be found from equations Equation (2.4.1) and Equation (2.4.2)

$$w_k(0) = \frac{\pi(1|1)g_k(1)}{1-\pi(1|1)} = g_k(1)\frac{\frac{\rho_k}{1+\rho_k}}{1-\frac{\rho_k}{1+\rho_k}} = g_k(1)\rho_k \tag{2.4.5}$$

Comparing Equation (2.4.5) to Equation (2.4.4), the Whittle index rule produces the optimal policy if we define ρ_k according to equation Equation (2.4.3).

The second justification for the ρ_k defined in Equation (2.4.3) is that the naïve index is obviously not capturing the system dynamics correctly when $\lambda(x_{K+1}+1) \gg \mu_k$. In this case, the naïve index makes assignments believing that future assets will be arriving for allocation rapidly and indefinitely; in reality the actual arrival rate in the closed-system is constrained by the finite number of assets in the system. The clever index addresses this issue by replacing the arrival rate Λ in the open-system not just with the repair rate as the naïve index does, but with a throttled version of the repair rate:

$$\Lambda \leftarrow \lambda(x_{K+1}+1)\frac{\mu_k}{\lambda(x_{K+1}+1)+\mu_k}. \tag{2.4.6}$$

We then define $\rho_k = \frac{\Lambda}{\mu_k}$ as in the open-system, which produces the desired quantity in Equation (2.4.3). We can view Equation (2.4.6) as stating the arrival rate to the

approximation system (Λ) is the repair rate ($\lambda(x_{K+1}+1)$) multiplied by a rough estimate of the fraction of time an asset is under repair, assuming an alternating renewal framework ($\frac{\mu_k}{\lambda(x_{K+1}+1)+\mu_k}$).

Regardless of how ρ_k is chosen, the procedure for using the Whittle index rule is as follows. Suppose the system is in state $\boldsymbol{x}$: the Whittle index rule does not keep a reserve, so the controller must wait until a repair completes to make a decision. We then proceed as follows:

Step 1: For each task k, note the number x_k of assets at that task;

Step 2: Set ρ_k, using either the naïve or clever approach described above.

Step 3: Calculate $w_k(x_k)$ using equation Equation (2.4.1), using ρ_k in the calculation of the stationary probabilities in Equation (2.4.2).

Step 4: Allocate the asset to the task with the highest calculated Whittle index $w_k(x_k)$; in case of ties choose the task first in the ordering.

2.4.2 Policies for Comparison

We shall compare the two Whittle index rule variants with three other policies: uniformly random allocation, greedy allocation, and where possible the optimal policy.

Uniformly random allocation is the simplest: whenever there are assets in the reserve, the controller allocates them randomly, with equal probability for each task. This

policy is not intended to be a sensible option, but instead acts as a basic benchmark: random allocation is what we might resort to if we knew nothing about the system state, and so any decent state-dependent policy should do better.

The second policy is the simple greedy policy: the controller never keeps a reserve, and whenever there is an asset to allocate, the controller puts it wherever results in the highest instantaneous reward rate. That is, the controller allocates the asset to task $l = \arg\max_k R(\boldsymbol{x} + \mathbf{e}_k)$. This policy is in many ways the simplest sensible policy, and if we did not have to worry about failures and replenishment would in fact be optimal. We examined other variants of the greedy policy (such as using $l = \arg\max_k R(\boldsymbol{x} + \mathbf{e}_k)/\mu_k$), but none were markedly different to the greedy policy and so we omit them.

The last policy we shall use for comparison is the optimal policy, which we compute by policy iteration ([24], section 8.5). We provide pseudocode for our policy iteration algorithm in Appendix B. As discussed earlier, the state-space grows very quickly, and as such we are only able to calculate the optimal policy for small values of N and K.

The optimal policy is the only policy we consider that makes use of the reserves; all other policies immediately assign an asset after repair. While we did consider allowing the other policies to retain reserves, there is no natural way to do this, and the performance of the clever Whittle policy is sufficient to make further finesses unnecessary.

2.5 Numerical Experiments

We now conduct numerical experiments to evaluate the performance of the policies defined above. We first re-parametrise the problem, so that the numerical experiments are clearer. We have five main inputs to our model: N, K, $g_k(x_k)$, $\lambda(x_k)$, and μ_k. We specify the reward functions $g_k(x_k)$ as follows. Recall that rewards are gained independently: $R(\boldsymbol{x}) = \sum_{k=1}^{K} g_k(x_k)$. In this section we focus on the following special form:

$$R(\boldsymbol{x}) = \sum_{k=1}^{K} \left(1 + \frac{Ak}{K}\right) g(x_k) \tag{2.5.1}$$

That is, we have one reward function $g(x)$ which is increasing and concave, and one parameter $A > 0$ that specifies how quickly reward rates increase with k.

We assume there is no queuing for repairs; as soon as an asset fails it immediately begins the repair process. Therefore, $\lambda(x) = lx$ and we normalise $l = 1$ throughout. The failure rates take the form $\mu_k = Mm_k$, where M is a constant we vary to control the magnitude of the failure rate, and the set $(m_1, m_2, ..., m_K)$ is a sequence of values near 1; the crucial aspect is the ratios of the m_k to each other. Specifically we look at four cases for the form of $(m_1, m_2, ..., m_K)$

Case 1: Constant m_k: $m_k = 1$ for all $1 \leq k \leq K$

Case 2: m_k increasing with k: $m_k = 0.5 + \frac{k-1}{K-1}$

Case 3: m_k decreasing with k: $m_k = 1.5 - \frac{k-1}{K-1}$

Case 4: m_k oscillating in k, $m_k = 1.5$ if k odd, $m_k = 0.5$ if k is even.

Given the discussion above, we require the following six parameters to completely specify our problem: $N, K, g(x), A, M, m_k$.

In our numerical experiments we examine all combinations of the parameters within the following limits:

- $N \in \{2, \ldots, 10\}$
- $K \in \{2, \ldots, 5\}$
- $g(x) \in \{1 - e^{-x/5}, \quad \log(1+x), \quad \sqrt{x}, \quad \min(x, 2)\}$
- $A \in \{1, 2, 3, 4\}$
- $M \in \{0.1, 0.5, 1, 1.5, 2, 3, 5, 10\}$
- m_k: the four cases described above

These parameter combinations produce 18432 different scenarios.

2.5.1 Policy Evaluation

Owing to the continuous-time Markov process structure of the problem, we can compute the long-term time-average reward exactly for a fixed policy $\boldsymbol{u}$. We denote the

long-term time-average reward $V^{\boldsymbol{u}}$ for policy $\boldsymbol{u}$. Any policy $\boldsymbol{u}$ induces a transition matrix $Q^{\boldsymbol{u}}$ on the state-space. We can find the stationary distribution $\pi^{\boldsymbol{u}}$ from the balance equations $\pi^{\boldsymbol{u}} Q^{\boldsymbol{u}} = 0$, with normalization constraint to ensure that $\pi^{\boldsymbol{u}}$ is a probability distribution. Once we have $\pi^{\boldsymbol{u}}$, we compute the long-term time-average reward via $V^{\boldsymbol{u}} = \sum_{\boldsymbol{x} \in \mathcal{X}} R(\boldsymbol{x}) \pi^{\boldsymbol{u}}(\boldsymbol{x})$.

2.5.2 Results

Our measure of performance is a policy's long-term time-average reward rate relative to the optimal, measured as the relative suboptimality gap of policy $\boldsymbol{u}$:

$$1 - \frac{V^{\boldsymbol{u}}}{V}$$

That is, for each set of parameters, we get a number between 0 and 1, which we report as a percentage, where 0% is identical to the optimal policy, and 100% equates to no reward at all. In the graphs, performance improves as we move up the y-axis.

Overall performance: The curves in Figure 3.5.1 show the performance quantiles across all 18432 parameter combinations. Table 3.5.1 gives a statistical summary for this case. Not surprisingly the uniformly random policy performs poorly relative to the other three. The clever Whittle index rule clearly dominates the other options, and is significantly more robust: even in the 5th percentile, it is at only 5.4% under-performance. The greedy policy performs better than the naïve Whittle index rule

across most percentiles, but both struggle significantly for some cases. The median performance is 0.6% for greedy, 1.3 % for naïve Whittle, and 0.4% for clever Whittle.

Failure rates: The failure rate has a significant impact on the results. Recall we use two parameters to specify the failure rate $\mu_k = Mm_k$. M controls the magnitude of the failure rates, whereas m_k determines the form of the rates in k. Table 2.5.2 presents the 5th percentile performance for each of the 4 cases for m_k (constant, increasing, decreasing, oscillating) and Figure 2.5.2 presents the quantile plots. Greedy performs reasonably well in Case 1 and Case 3 but not well in Case 2 and even worse in Case 4. The naïve Whittle index rule slightly dominates over greedy in Case 1 (even over the clever Whittle), and very significantly across lower quantiles in Case 4, but has a significantly worse performance in Case 2. Clever Whittle performs consistently well for all 4 cases, showing it is robust to changes in forms of failure rates.

We see similar behaviour in Table 2.5.3 and Figure 2.5.3 as we vary the magnitude M of the failure rates. Recall the repair rate has the form $\lambda(x) = x$, so $M = 10$ ($M = 0.1$) corresponds to very large (small) failure rates relative to repair. For large M, greedy performs very well and naïve Whittle performs almost as poorly as the uniformly random policy. In these cases, assets spend most of the being repaired, and we rarely have multiple assets allocated. As such, being myopic is a reasonable choice.

For small M when assets fail slowly compared to repairs, the greedy performs almost as poorly as the uniformly random policy and the naïve Whittle performs well, even

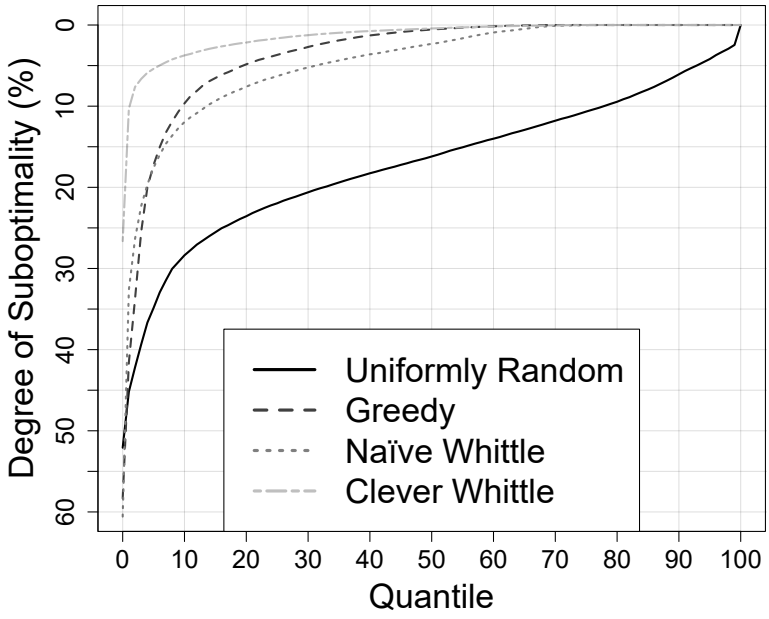


Figure 2.5.1: Policy performance for each quantile.

Policy	Mean	Min.	5^{th} Percentile	Lower Quartile	Median	Upper Quartile	Max.
Uniformly Random	17.1	52.1	34.1	22.0	16.2	10.7	0.0
Greedy	3.6	58.3	18.4	3.7	0.5	0.0	0.0
Naïve Whittle	4.6	60.6	14.5	6.3	2.3	0.0	0.0
Clever Whittle	1.3	26.6	5.4	1.6	0.4	0.0	0.0

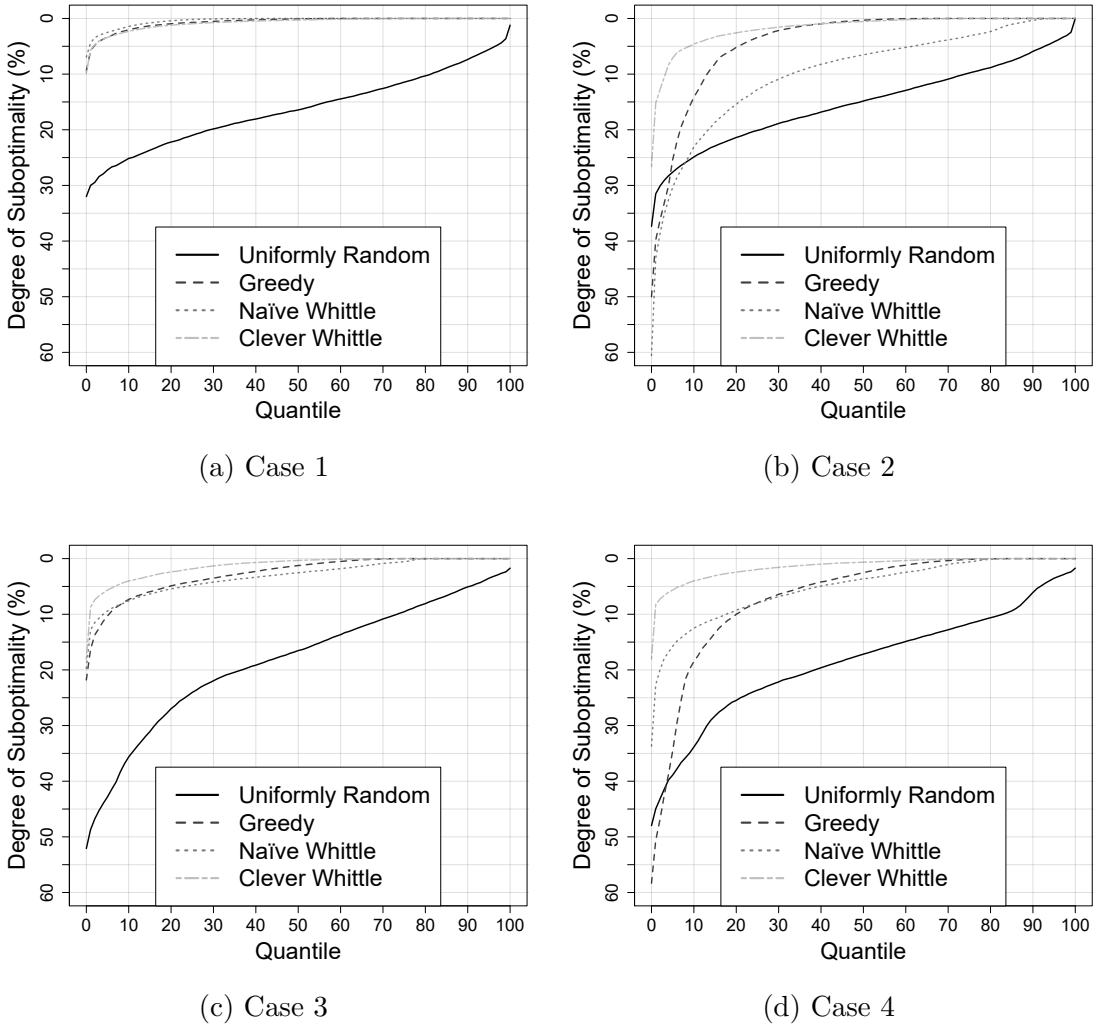
Table 2.5.1: Mean and five-number summary given as the relative suboptimality gap (in percent).

Policy | Case	Case 1 (constant)	Case 2 (increasing)	Case 3 (decreasing)	Case 4 (oscillating)
Uniformly Random	27.2	27.6	42.8	38.9
Greedy	3.4	25.4	10.3	34.5
Naïve Whittle	2.5	30.3	9.4	15.7
Clever Whittle	3.4	7.1	5.6	5.6

Table 2.5.2: The 5th percentile for each policy, stratified by the failure rate forms.

Policy $\|M$	10	2	1.43	1	0.5	0.33	0.2	0.1
Uniformly Random	28.6	26.9	25.8	25.1	26.4	28.9	32.1	46.6
Greedy	2.2	3.1	3.6	4.3	8.1	12.3	17.3	45.7
Naïve Whittle	27.0	24.5	21.5	19.4	14.0	11.0	8.0	5.0
Clever Whittle	4.2	4.3	4.6	4.5	4.5	5.1	6.0	11.2

Table 2.5.3: The 5th percentile for each policy, stratified by the failure rate magnitude M.



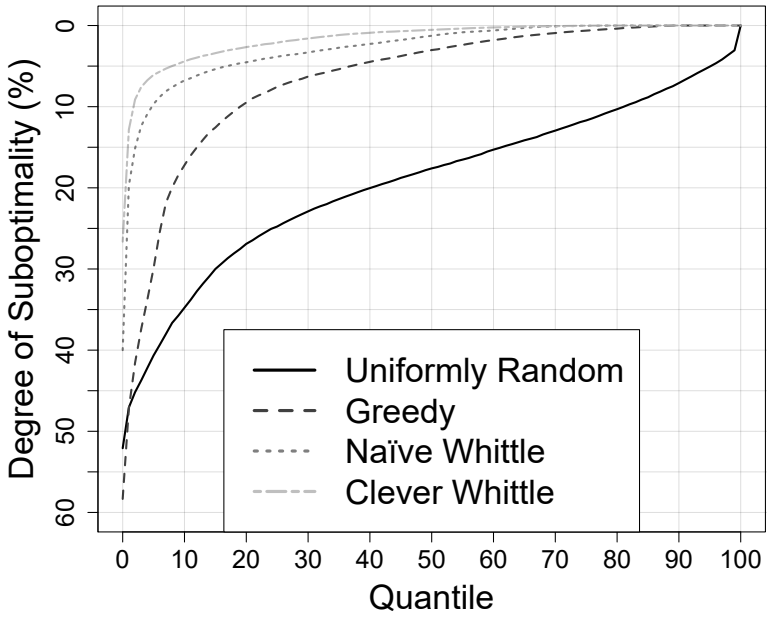
(a) Case 1

(b) Case 2

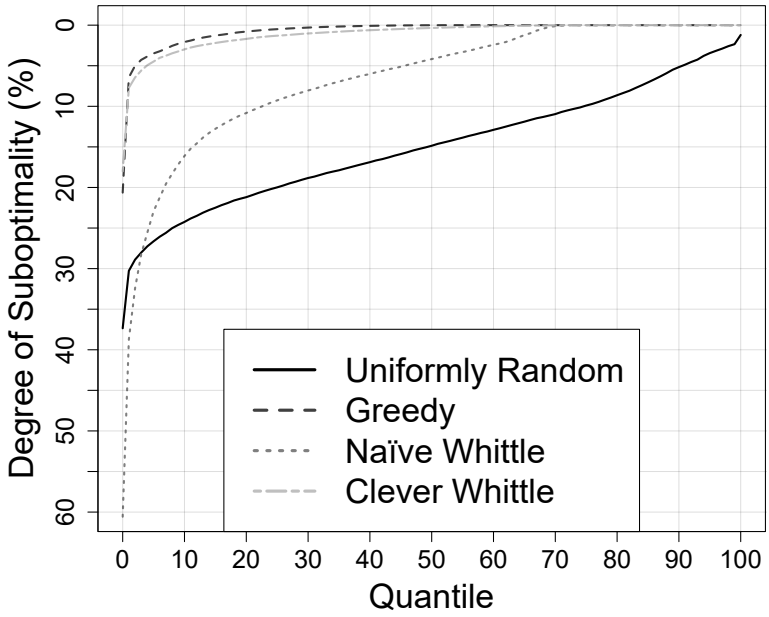
(c) Case 3

(d) Case 4

Figure 2.5.2: Quantile plots, stratified by the failure rate forms.



(a) $M \leq 1$



(b) $M > 1$

Figure 2.5.3: Quantile plots, stratified by the failure rate magnitude M.

better than the clever Whittle. In this case the rapid barrage of repairs produces a system that more closely mimics the open-system approximation with a constant arrival rate. Although the clever Whittle does not perform as well as either greedy or naïve Whittle in the extremes, it is again very robust across all values of M.

Use of the reserve: The reserve is used by the optimal policy in 33% of the scenarios. In the most extreme cases, the system is in a state with an asset in reserve 25% of the time. However, overall the reserve component usually plays only a very minor role. Only 2% of the scenarios that optimally use the reserve are in a reserve state more than 10% of the time. The fact that our policies, which do not use reserves, perform so well illustrates that in general, the reserve does not serve a critical role.

The reserve is more important for larger N and smaller K. The reserve is also used more frequently for larger M (greater failure rate relative to repair rate). In this case a decision maker might prefer to wait for another imminent failure before allocating an available asset to ensure the allocation is effective, because the next allocation opportunity may not be for a while.

Other Points: For the four reward function options, $g(x) = \min(x, 2)$ most often produces the worst performance. This is not surprising as none of our policies allows for reserves, and clearly it is sub-optimal to allocate more than two assets per task for this reward function. Consequently performance degrades for large N and small K for this reward function.

Finally, there is a slight degradation in performance of the policies as we increase the

number of assets N. For the median, the clever Whittle's median relative suboptimality gap is 0.0% at $N = 2$ and increases to 0.9% at $N = 10$. The corresponding degradation at the 5th percentile for the clever Whittle index rule is from 2.2% to 6.4%.

2.6 Conclusion

We have formulated a model for the allocation of assets when the assets are subject to failure. We established a solid theoretical grounding for the Whittle index rule and examine the performance of two Whittle index rule variants along with several other policies. From the numerical experiments, the clever Whittle index rule performs well and appears to be a reasonable and robust policy to generally use in practice. Additionally it is computationally simple, and extremely quick compared to using policy improvement to calculate the optimal policy.

There are several avenues for future research to improve our model. Rewards may not accumulate independently across the tasks. One might need a critical mass of assets at each task before the controller can gain positive rewards. The restless bandit model would not apply to a more complex reward function with dependencies across tasks; both the Whittle index rule and the greedy policy would need to be modified. We consider this in the next chapter.

The Markovian assumptions, while unrealistic for some real-world applications, are

not easy to discard. In practice it may be better to model failure times as uniform or deterministic: each asset has a certain level of charge, which depletes in a fairly predictable way. A possible intermediate approach would be the use of Erlang distributions, which are still simple enough to allow for effective calculations, while sufficiently complicated to model actual situations. One could also perform a simulation exercise to apply the policies formulated here for the Markovian case to a non-Markovian setting.

There might be delays between assignment and when an assets starts performing a task, such as travel times. Some of this delay can be absorbed into the repair process. We could also modify the reward function to account for the delay cost.

We mentioned in Section 2.4.2 the possibility of adding a reserve to the policies derived from the restless bandit model. This would best be done by adding a reserve to the restless bandit model, although there is no simple way of doing this. In particular the presence of a reserve links the different tasks, something wholly incompatible with a restless bandit model.

The assumption that assets cannot be moved from one task to another may be reasonable in certain situations in practice (e.g. where the tasks are located too far away from each other or where there are few specialized assets), but the range of potential applications would greatly expand if asset relocation was allowed, possibly incorporating a switching cost and/or delay. Whittle indicies for such models have not yet been studied, only its special case the Gittins index, which does not apply in this

context.

One last complication is rewards that are unknown *a priori*. The controller would have to learn about the rewards gained per state as she tries to stay in high-reward states. The framework here is analogous to a classic bandit problem [28]. Unlike a bandit problem, we have a complicated state-space, and the dynamics are not entirely under our control. This would be particularly relevant to environmental monitoring cases, where the controller might not initially know which locations are most important to observe.

Chapter 3: The Dependent Case

In this chapter, we address a particular case in which we are required to form a chain of assets to communicate any discoveries back to base. We first motivate the problem, then formulate it as a continuous-time, average-reward Markov decision problem (MDP) with impulsive controls. We identify several features necessary for a policy to be optimal, and use this to formulate a variety of heuristic policies.

We conduct extensive numerical experiments, and determine that a simple policy enhanced by a single round of policy iteration provides the best balance between computational cost and effectiveness, equalling the optimal policy 75% of the time, and performing within 10% of optimal 95% of the time. We additionally extend to cases with multiple reward-gaining tasks, multiple phases, or with non-Markovian dynamics, and find the same policy is similarly effective there, performing within 10% of optimal 85% and 95% of the time for the first two cases respectively, and performing better than other policies in the non-Markovian case.

3.1 Introduction

3.1.1 Problem Description

In this chapter we consider a problem in which a number of interchangeable assets need to be allocated to a number of different tasks.

The inspiration for this was a problem of searching with communications arising in a naval context. In this problem, we are required to search for targets in an area of ocean, and any detections must be reported back to some base. The communications are comparatively short-range, and so it is necessary to assign some of the assets to act as relays to ensure the transmissions are properly received.

The tasks are split into two categories: communications and search. We require a certain threshold of assets communicating before any reward can be gained from assets allocated to search. The objective is to find a way of allocating assets to tasks that maximises the long-term time-average reward rates obtained.

The overall problem is motivated by the same problems as in [1], including those of UAV search and patrol [2, 20, 94, 101, 102, 103], assignment problems [104], search and rescue work [92, 93] and environmental monitoring [94].

The reward functions in this chapter are all 'dependent' in some manner: the rewards are not gained independently from each task, but instead depend on the overall state

of the system. In particular, we focus on cases where we are required not merely to allocate assets to gain reward, but also to communicate or transfer said reward.

3.1.2 Chapter Structure

We begin in Section 3.2 by defining the problem as a Markov decision process (MDP). In Section 3.3 we give two results justifying the use of a particular form of policy, and explain more fully how to use and improve on this. We then describe in Section 3.4 how we propose to numerically test these policies, and give the results in Section 3.5. We then move on to two extended cases in Section 3.6 and Section 3.7, and then the non-Markovian case in Section 3.8. Finally, in Section 3.9 we talk about possible extensions of this work.

3.2 Mathematical Formulation

We now describe our first formulation of the problem; there will be others later. We treat this problem as an MDP initially, even though this requires several unrealistic assumptions to be made; later sections will relax some of these conditions. Everything in this section except the reward rates is as in Section 2.2, so we will not describe those matters.

Since the actions move the state instantaneously, for our purposes it is sufficient to consider reward rates that are independent of actions. We write that in state x, we

gain reward at a rate $R(x)$. The rewards are gained in the following form:

$$R(x) = g(x_K) \prod_{k=1}^{K-1} I(x_k > 0) \tag{3.2.1}$$

We need at least one asset in each of the first $K-1$ tasks, and only then do we get reward, depending on the number of assets at task K. We impose that g is increasing and that $g(0) = 0$. These conditions are both reasonable: adding more assets should give more reward, and no assets should give no reward. There is no reward gained for assets under repair or in reserve, again a reasonable condition.

There is a cleverer equivalent formulation of this problem, as long as all the first $K-1$ failure rates (the μ_k) are the same. Set $K = 2$ and define C as the number of assets required for coherent communications; for the model just before, $C = K - 1$. Use the reward function:

$$R(x) = I(x_1 \geq C)g(x_2) \tag{3.2.2}$$

A brief argument now justifies why the two formulations are equivalent, given that the first $K-1$ failure rates are identical:

1. As the first $K-1$ of the μ_k are of equal value, and the system is Markovian and hence memoryless, the state of the first $K-1$ tasks can be summarised entirely by how many of the tasks have an asset allocated to them.

2. As the reward function gives no benefit to having more than one asset allocated to any of the first $K-1$ tasks, and we can allocate from the reserve at the instant of any failure, following any sensible policy, we have $x_k \leq 1 \forall k \leq K-1$.

3. Therefore, $\prod_{k=1}^{K-1} I(x_k > 0) = 1$ iff $x_k = 1 \forall k \leq K-1$.

4. This second condition is equivalent to $\sum_{k=1}^{K-1} x_k = K-1$.

5. The search task is exactly the same in both formulations.

6. Therefore the two formulations are equivalent, under the conditions given, for any sensible policy.

This two-task formulation is neater, simpler, and gives rise to a smaller state-space, allowing efficient computation of the optimal policy for larger problem instances, and we therefore use it in the rest of this chapter.

3.3 Policies

In this section we describe the various policies we have formulated for this problem, starting with the framework common to them all.

3.3.1 The Policy Skeleton

Utilising the two-task formulation makes the state-space smaller, and so the problem more tractable. As such, we use the two-task formulation until mentioned otherwise.

That being said, it is still not possible to compute the optimal policy for relatively small values of N: having even 15 assets causes serious computational problems. Recall that the statespace has size $\begin{pmatrix} N+K+1 \\ K+1 \end{pmatrix} = \frac{(N+K+1)!}{(K+1)!N!}$, so that in our case $|X| = \frac{(N+4)!}{6*N!} = O(N^4)$. This does not look too bad, but policy improvement takes per-step a time $O(|X|^3)$

It is therefore necessary to produce policies that can actually be used in practice. We start with a structure that captures the essential logic common to any sensible policy; this is depicted with algebra in Figure 3.3.1 and without the algebraic expressions in Figure 3.3.2. We also present two results to justify some of the diagrams' logic.

Result #1 (boxes 2 and 3): $x_1 \leq C$ for any sensible policy, at all times, so that we never have more that C assets at the communications task.

Restatement: Given a policy u that can ever allocate assets such that $x_1 > C$, the policy u^* that follows u except to reserve assets whenever it would make $x_1 > C$ has no worse a long-term average-reward.

Proof: Firstly, u^* always has at least as many assets reserved at all times as u, because reserving assets that are allocated by u is the only difference. So at any point, we can go freely from the states we reach via u^* to those we would reach using u, by allocating the assets to task 1.

Consider the Bellman equation 2.2.3. The important part here is that everything on the RHS is ensconced within the $\max_{a \in \mathcal{A}(x)}$. Therefore, if we replace the set $\mathcal{A}(x)$ by

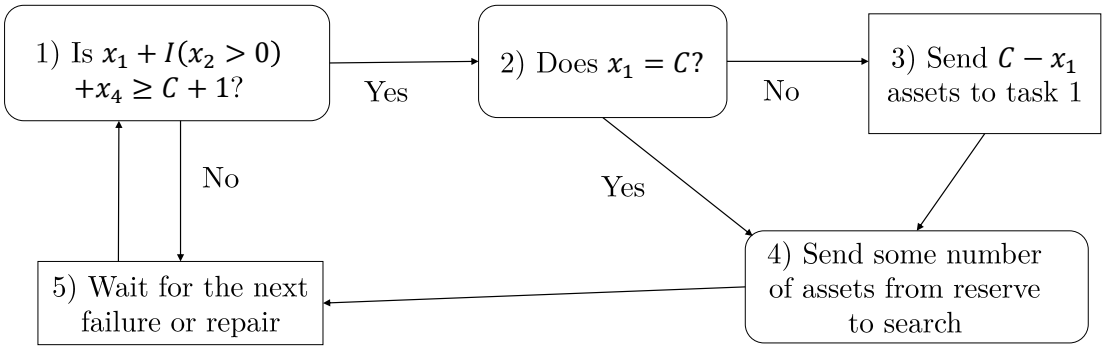


Figure 3.3.1: Flowchart detailing in algebra the logic of any sensible policy.

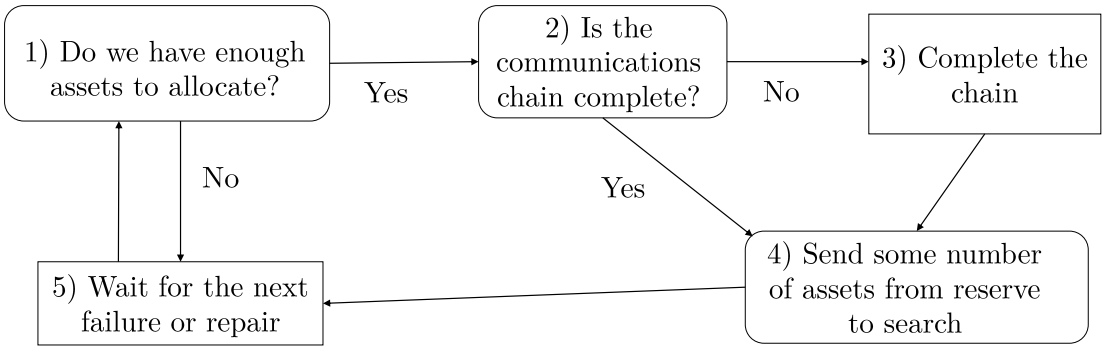


Figure 3.3.2: Flowchart detailing in words the logic of any sensible policy.

a set $\mathcal{A}'(x)$ such that $\mathcal{A}(x) \subseteq \mathcal{A}'(x)$, we can only get a larger value on the LHS.

As such, the resultant average reward rate can only be larger, and so the policy can only have been an improvement over the original one. But this is exactly the relation of u and u^*, so u^* must be at least as good as u, as required.

Result #2 (box 1): $x_1 + I(x_2 > 0) + x_4 < C + 1$ iff no allocation of assets can get us reward.

Proof: We need $x_1 \geq C, x_2 \geq 1$ in order to get reward, as $g(0) = 0$, so we do need $x_1 + I(x_2 > 0) + x_4$ to be at least $C + 1$.

As proven above, we should never reach $x_1 > C$, so if $x_1 + I(x_2 > 0) + x_4 \geq C + 1$, we must have $x_4 \geq C - x_1$ if $x_2 > 0$, and $x_4 \geq C + 1 - x_1$ if $x_2 = 0$. So if $x_1 + I(x_2 > 0) + x_4 \geq C + 1$, we can always allocate assets to reach a reward-gaining state.

3.3.2 θ-Policies

The diagram above does not specify a policy: to create one, we need to decide on what happens in box 4. The first approach we take is to attempt to keep a constant split of assets between task 2 and task 4 - between searching and the reserve. We control the split with a parameter $\theta \in [0, 1]$, as follows:

1. Arrive at box 4. We are in some state $x = (C, x_2, x_3, x_4)$. $x_4 > 0$ as otherwise we have no decision to make;

2. If $x_2 = 0$, send one asset to task 2 from the reserve;

3. We want to have a fraction θ of the deployable assets at task 2. That is, we start with x_2 assets at task 2 and x_4 at task 4, and want to end up with $x_2/(x_2 + x_4) \approx \theta$. So we send $\lceil \theta(x_4 + x_2) - x_2 \rceil$ assets* to task 2 from the reserve, or none if that expression is negative;

4. Move on to box 5.

For this set of policies, θ controls how conservative the policy is. For low values of θ, we try to keep most of our assets back, while for θ closer to 1 the policy is aggressive and sends most of the free assets to task 2, to search.

3.3.3 Optimising, and Policy Improvement

The θ policies described above form a one-parameter family of policies, so it is natural to ask if we can choose the best value of θ. The optimal value depends on the precise details of the problem, but it is not exceptionally difficult to find. We have one parameter θ, a small state-space $(0, 1]$ to search, and the function evaluations are not too difficult: we just find the resultant control $u(\theta)$, find the stationary distribution $\pi^{u(\theta)}$ (we explain how to do this in section Section 1.2.1 and Appendix B.2), and then the average reward rate is just $\sum_{x \in \mathcal{X}} R(x)\pi^{u(\theta)}(x)$.

This gives us one policy: the optimal θ policy. Another approach that we could

*The ceiling $\lceil \rceil$ is necessary because assets are discrete; we choose ceiling instead of floor to encourage allocation.

apply to any policy but here use only on θ policies, is to apply a single step of policy improvement [24] to an existing policy. This is guaranteed to give a better policy, and has a fixed runtime, as opposed to full policy improvement which in the worst case could take $2^{|\mathcal{X}|}$ steps [26].

At this point, we might reasonably ask if there are any sensible families of policies for which we could analytically compute the stationary distribution π. If this were the case, we could then optimise over the parameters of those families just like we can over θ. The answer to this is sadly no, for several reasons.

The first difficulty is that π need not exist. Consider a policy that never reserves assets if $x_{K+2} = 0$, but if it has more than one asset reserved, keeps at least two reserved for the rest of time. This policy has essentially two different stationary distributions, on two communicating classes of states. This difficulty can be avoided by choosing less silly policies; a more intractable problem is that the overall dynamics depend in unpredictable ways on the decisions made in different states.

Additionally the size of the statespace depends on the parameter N, and K in the full formulation. Taken together, these difficulties render it impossible to find the stationary distribution analytically.

3.3.4 The Concavity Policy

The θ policy as promulgated above uses the same θ for all states. Approaching the problem of choosing θ from a different angle, if $g(x)$ is linear or close to linear, a

higher θ would be more appropriate, while if $g(x) = \min(x, 1)$ for example, then we should never send more than one asset to task 2. The concavity policy uses a measure of how concave g is to pick an appropriate θ, given the current state x.

The two functions that guided us whilst formulating this policy were $g(x) = x$ and $g(x) = \min(x, 1)$. For the first one, we want θ either at 1 or close to it; for the latter, we want $\theta = 1/(x_2 + x_4)$, so that we only send one asset to task 2.

The expression we use starts with the following formula, which is the ratio of the area beneath a function $f(x)$ satisfying $f(0) = 0$ and the area beneath the line tangent to f at $x = 0$, as displayed in Figure 3.3.3:

$$\theta(x) = \frac{\int_0^x f(t)\,dt}{f'(0)\int_0^x t\,dt} \tag{3.3.1}$$

Now, our g are only defined on the integers, so we need to substitute expressions for $g'(0)$ and for the integrals. The standard approximation for the derivative of a function f at x is $f'(x) = \frac{f(y)-f(x)}{y-x}$; because $g(0) = 0$, we can approximate $g'(0) = \frac{g(1)-g(0)}{1-0} = g(1)$.

For the integrals, we can evaluate them using the trapezium rule, which is exact for the integral in the denominator. This yields:

$$\theta(x) = \frac{1}{g(1)}\frac{2g(1) + 2g(2) + \ldots + 2g(x-1) + g(x)}{x^2} \tag{3.3.2}$$

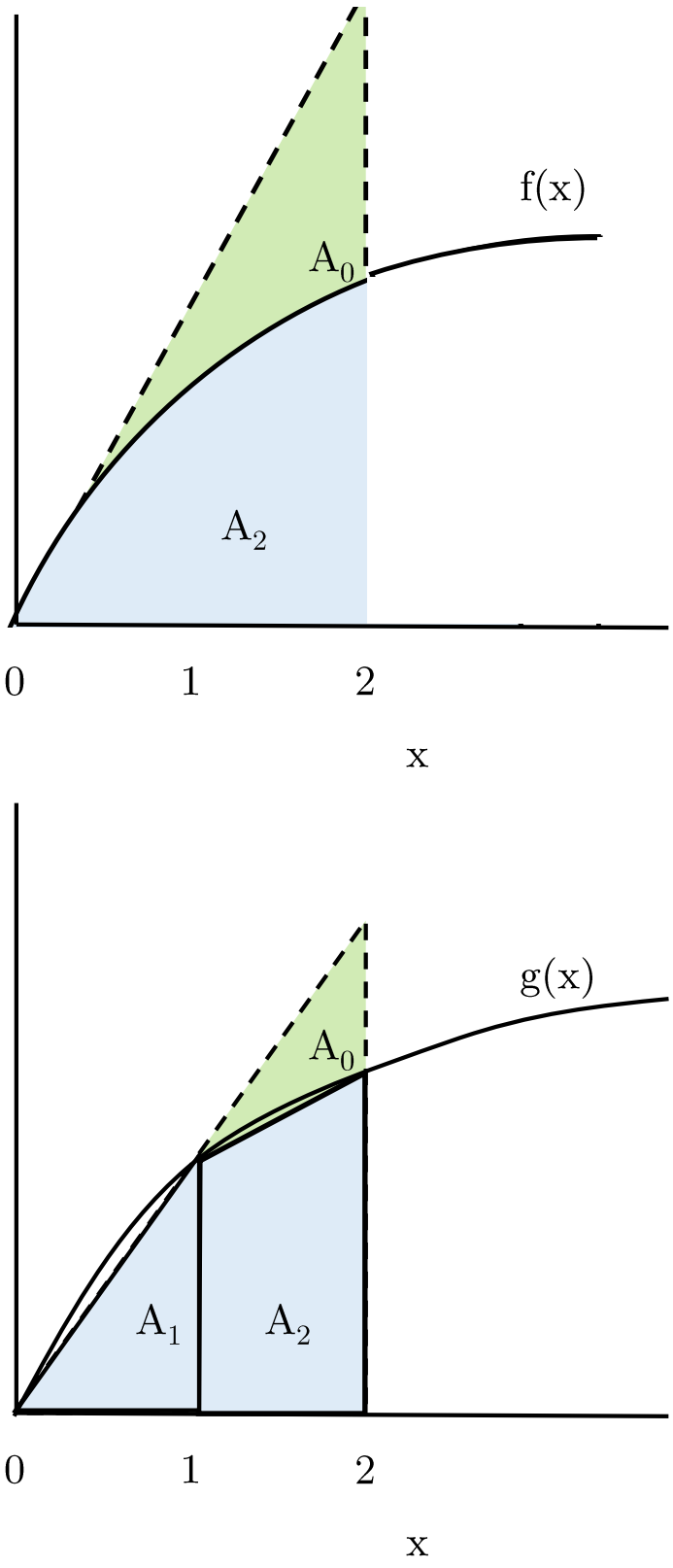


Figure 3.3.3: The diagram on the top corresponds to Equation (3.3.1), while the diagram below corresponds to Equation (3.3.2). In each case, the area in blue corresponds to the denominator and the sum of the areas in green and blue to the numerator.

To use this, we follow the θ policy, but every time we come to check if we should allocate assets to task 2 from the reserve, we use $\theta(x_2)$, allocating assets one at a time and recalculating $\theta(x_2)$ each time.

3.4 Procedure for Experiments

We now carry out some detailed numerical experiments to determine which of the policies above performs best.

3.4.1 Procedure & Parameter Values

Because of the Markovian structure of the problem, we can evaluate the infinite horizon time-averaged reward of a given policy exactly, as in Section 1.2.1 and Appendix B.2.

We also need to specify the parameter values for the problem. Given that we are using the two-task formulation, we need the following parameters to specify the problem:

$$\mu_1, \mu_2, \lambda(x), N, C, g(x) \tag{3.4.1}$$

There is a subtle redundancy here, as we had before in Section 2.5. If we multiply μ_1, μ_2 and $\lambda(x)$ by the same positive constant, the system speeds up, but the stationary distribution and hence the long-term average reward rates for any policy remain

unchanged. As such, we vary μ_1 and μ_2, and fix $\lambda(x) = x$; this form ensures a certain amount of reliability in the system.

We vary the remaining parameters as follows:

- $N = 3, \ldots, 8$; N must be at least 3 to give an interesting problem, and we run to as high a value as computational time allows
- $C = 1, \ldots, 8$, but with $C \leq N - 2$ so the problem is interesting.
- $\mu_1, \mu_2 \in \{0.1, 0.5, 0.7, 1, 2, 3, 5, 10\}$; each failure rate seperately takes each value in the set
- $g(x) \in \{1 - e^{-x/3}; \min(x, 2); \log(1 + x); \sqrt{x}; x\}$; these reward functions give a good range of behaviours, all of which are increasing and start at 0. Both of these are reasonable conditions: more assets should help, and having no assets would give no reward.

This comes out to 6720 different combinations of the parameters.

3.4.2 Policies

We investigate the following six policies in comparison to the optimal policy:

1. Uniform random allocation: whenever there are assets in the reserve, the controller allocates them randomly, with equal probability for each task. This policy

is not intended to be a sensible option, but instead acts as a basic benchmark: random allocation is what we might resort to if we knew nothing about the system state, and so any decent policy should do better.

2. The θ policy, with $\theta = 0.5$.

3. The $\theta = 0.5$ policy, improved by one stage of policy improvement, which we call $\theta+$ or $\theta = 0.5+$. We described policy improvement in Section 1.2.1 and Appendix B.1.

4. The optimal-θ policy; that is, the θ-policy with the best possible θ for the problem at hand. This can be done quite simply, as we are searching an interval $(0, 1]$, and we do not have to be too precise, as the discreteness of assets means small changes in θ may not change the policy at all.

5. The optimal-θ policy, improved by one stage of policy improvement, which we call optimal-$\theta+$*.

6. The concavity policy, as laid out in Section 3.3.4 above.

We are limited in the number of scenarios for which we can evaluate these policies by computational time, and making the problem larger does not make it particularly more interesting.

*There is an ambiguity in this: it could mean optimal of $(\theta+)$, but we mean (optimal-θ)+. This is English's fault, and unavoidable.

3.5 Numerical Results

In this section we present the results of our numerical experiments. Our measure of the performance of a policy u is its long-term time-average reward rate relative to the optimal, measured as a fraction:

$$\frac{V^{\boldsymbol{u}}}{V^{opt}}$$

That is, for each set of parameters, for each policy we get a number between 0 and 1, where 1 is a reward-rate identical to that of the optimal policy, and 0 equates to no reward at all. In the graphs, performance improves as we move up the y-axis.

Our first figure, Figure 3.5.1 shows the quantiles of the relative reward, for each of the policies above; summary statistics are given in Table 3.5.1.

It is interesting to look at the results, stratified by the form of $g(x)$ in Table 3.5.2, and by N in Table 3.5.3. We use the 25^{th} quantile because the median is too high to clearly show any differences.

We are also interested in the relative time-complexities of the different policies; the times taken for each policy to be computed are shown in Figure 3.5.2 as box-plots across all parameter values, with the y-axis as a log scale. Theoretically, we would expect a stage of policy improvement to take a time $O(|\mathcal{X}|^3)$, and for each policy evaluation for the optimal-θ policy to have the same time complexity.

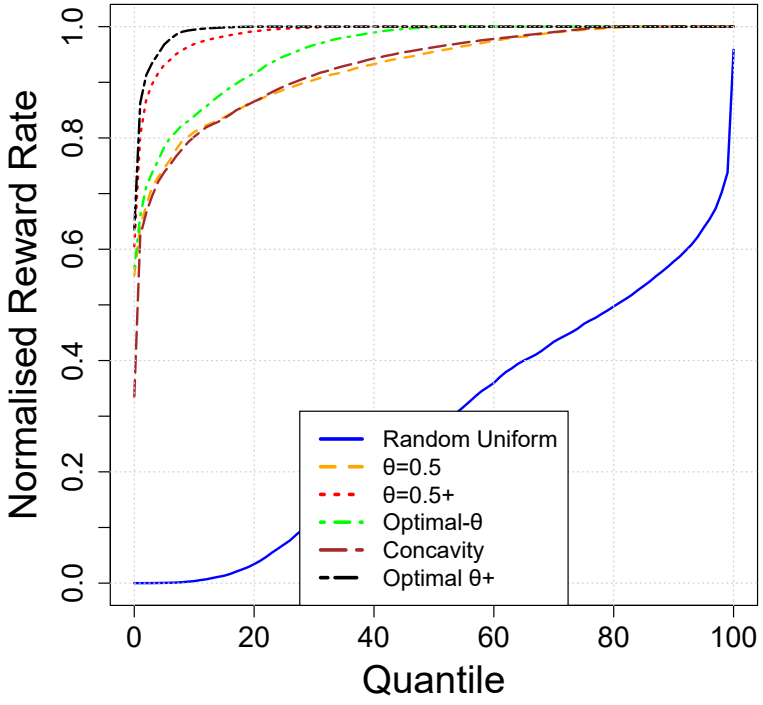


Figure 3.5.1: Policy performance for each quantile.

Policy	Minimum	First Quartile	Median	Mean	Third Quartile	Maximum
Random Uniform	0.00	0.07	0.27	0.28	0.47	0.96
$\theta = 0.5$	0.55	0.89	0.95	0.93	0.99	1.00
$\theta = 0.5+$	0.61	1.00	1.00	0.99	1.00	1.00
Optimal θ	0.57	0.95	1.00	0.96	1.00	1.00
Optimal $\theta+$	0.64	1.00	1.00	0.99	1.00	1.00
Concavity	0.33	0.89	0.96	0.93	1.00	1.00

Table 3.5.1: Mean and five-number summary for our six policies.

Policy	Linear	Logarithmic	Minimum	Negative-Exponential	Square Root
Random Uniform	0.073	0.069	0.068	0.070	0.067
$\theta = 0.5$	0.826	0.917	0.877	0.898	0.933
$\theta = 0.5+$	0.979	0.999	0.992	0.996	1.000
Optimal θ	0.932	0.962	0.922	0.956	0.968
Optimal $\theta+$	1.000	1.000	0.999	1.000	1.000
Concavity	0.859	0.898	0.934	0.901	0.885

Table 3.5.2: 25^{th} quantile of the normalised reward rate, stratified by form of $g(x)$.

The graphed results are in line with our expectations. Random uniform is fast (it does no computation), and so is $\theta = 0.5$, which only needs to calculate two expressions. Concavity is next, as it requires a few computations, but no linear algebra. $\theta = 0.5+$ is next quickest, as it only requires a single matrix solve, while optimal-θ takes several, and optimal-$\theta+$ takes one more. Lastly, calculating the optimal policy takes the longest, as it requires many matrix solves, and some more operations to choose the new policy at each stage.

3.5.1 Summary of Results

A step of policy improvement is key: the two best performing policies are $\theta = 0.5+$ and optimal-$\theta+$. Taking into account both computational time and experimental effectiveness, the best policy is the optimal-$\theta+$ policy, improved by a step of policy improvement. The $\theta = 0.5+$ policy performs slightly worse, but is a little quicker to compute, with the optimal-θ policy only slightly behind.

We can get more detail by looking at the stratified results. First of all, the form $g(x) = \min(x, 2)$ is the most challenging of the reward functions; this should not surprise us, as it is the most sharply changing. Apart from that, the reward rates for each of the reward functions are largely in line with the averages overall.

Looking at Table 3.5.3, we have a similar pattern. The problem gets harder for larger N, with no particularly interesting features other than that.

Lastly, which case did optimal-$\theta+$ do worst on? The answer was the problem deter-

Policy	3	4	5	6	7	8
Random Uniform	0.317	0.190	0.117	0.075	0.037	0.018
$\theta = 0.5$	0.948	0.952	0.930	0.900	0.867	0.829
$\theta = 0.5+$	1.000	1.000	1.000	0.998	0.992	0.987
Optimal θ	1.000	1.000	0.985	0.947	0.902	0.855
Optimal $\theta+$	1.000	1.000	1.000	1.000	1.000	0.998
Concavity	0.982	0.977	0.948	0.907	0.858	0.816

Table 3.5.3: The 25^{th} quantile of the normalised reward rate, stratified by N.

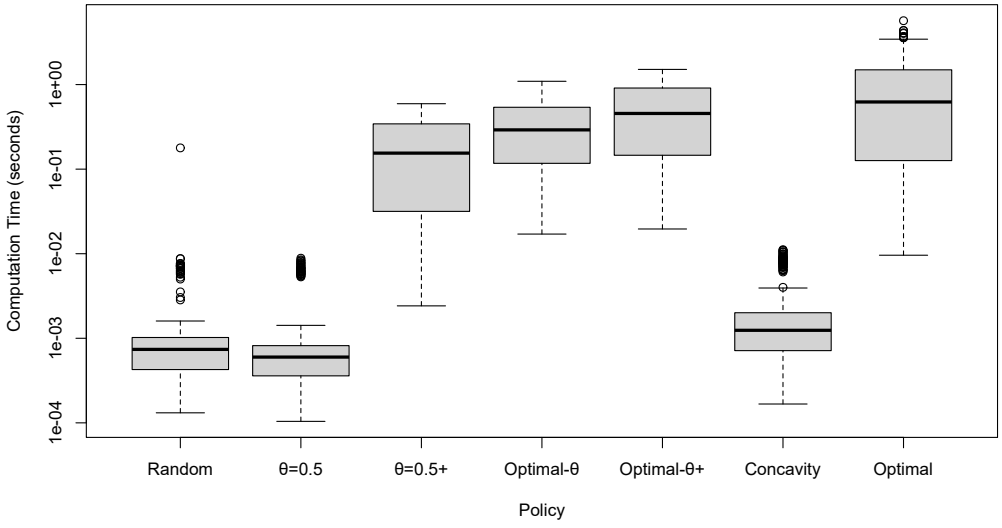


Figure 3.5.2: Computation time for each policy, box-plotted across all scenarios. The y-axis shows the time taken to compute the policy, on a logarithmic scale.

mined by the following parameters: $N = 8, C = 5, \mu_1 = 10, \mu_2 = 1, g(x) = min(x, 2)$. This is clearly a nasty case: the μ_k are significantly different, N is large and has a gap to C, and $g(x)$ has the most difficult form.

Figure 3.5.2 shows the distributions of the times taken to compute the various policies, and is what we would expect. Random uniform, $\theta = 0.5$ and concavity are all quick; θ-+, optimal-θ and optimal-θ+ are all slower, and the optimal policy is the slowest to compute of all the policies. While we would have predicted this ordering on theoretical grounds, it is helpful to have experimental verification.

3.6 An Extended Case

The model given above is extremely specific. We make it more general by starting with the same general framework, and extending the search tasks. Instead of only having one, we have $K - 1$ for some integer $K \geq 2$, giving $K + 2$ tasks in total, writing the state of the repair process as x_{K+1} and of the reserve as x_{K+2}. We keep the assumption about all the communications tasks' failure rates being the same, and so can collapse communications to task 1.

Now, we have a reward function g_k for each search task, and the overall reward function is:

$$R(x) = I(x_1 \geq C) \sum_{k=2}^{K} g_k(x_k) \tag{3.6.1}$$

We are still keeping everything else the same: failures and repairs are still Markovian, and we are still interested in the long-term average reward gained, so that Equation (2.2.3) still applies. See Figure 3.6.1 for the dynamics.

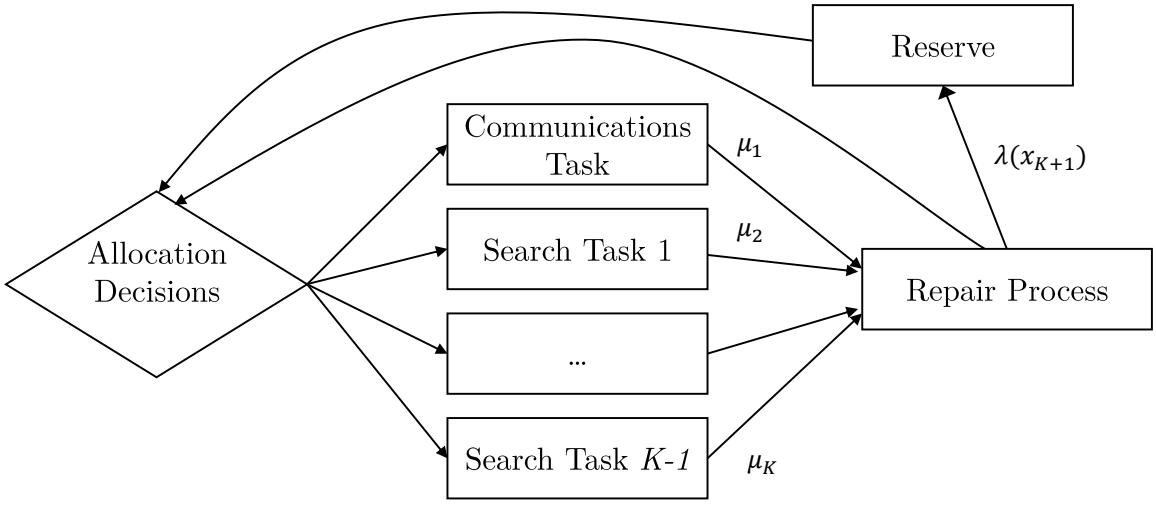


Figure 3.6.1: Flowchart detailing the dynamics of the full-K problem.

We can use the previous work for this case: we can still determine the balance between allocation and reserving as before, and follow the logic in Figure 3.3.1. The difference is that once we choose to allocate an asset to searching, we must then choose which of the search tasks $2, \ldots, K$ we wish to allocate to.

In Chapter 2, we addressed the problem of allocating between K independent tasks, in a similar MDP framework. The conclusion from that work was to use a Whittle index policy. In essence, the Whittle index policy decomposes the problem into K tasks, and quantifies how much each task would 'pay' to have an additional asset sent its way. This is done using a restless bandit model of the problem.

We use the Whittle index policy as follows:

Step 1: For each task k, note the number x_k of assets at that task;

Step 2: Calculate $w_k(x_k)$ as below.

Step 3: Allocate the asset to the task with the highest calculated Whittle index $w_k(x_k)$; in case of ties choose the task with the smallest numerical identifier. If we have multiple assets to allocate, go back to step 1.

We determine the index $w_k(x_k)$ for task k as follows:

1. Define the effective arrival rate as $\Lambda_k = \frac{\lambda(x_{K+1}+1)}{\lambda(x_{K+1}+1)+\mu_k}$, and write $\rho_k = \Lambda_k/\mu_k$
2. Calculate the stationary distribution probabilities $\pi(x|x_k) = \frac{\rho_k^x/x!}{\sum_{y=0}^{x_k}\rho_k^y/y!}$.
3. The Whittle index for task k is then:

$$w_k(x_k) = \frac{\sum_{x=0}^{x_k+1}\pi(x|x_k+1)g_k(x) - \sum_{x=0}^{x_k}\pi(x|x_k)g(x)}{\pi(x_k|x_k) - \pi(x_k+1|x_k+1)} \tag{3.6.2}$$

We use this form for the Whittle index because in Chapter 2 it gave the best experimental results.

We can combine this with the policy skeleton by using the Whittle indices to decide allocation to the search tasks: whenever we decide to allocate to searching, allocate to the search task with the highest Whittle index. If we have multiple assets to allocate, we do so sequentially: we allocate one asset, reevaluate the Whittle indices, and iterate until all assets we have decided to allocate have been allocated.

Figure 3.6.2 shows a comparison between random uniform allocation which is again serving as a lower bound of sorts, the policy produced from joining the $\theta = 0.5$ policy with the Whittle index policy, and that policy improved by one step of policy improvement. The plot shows quantiles of the reward, as a fraction of the total reward, over the same parameter values as before, with μ_2 taken as the failure rate for all search tasks, and K being varied between 2 and 5. Summary statistics are given in Table 3.6.1.

We can see that, while our best policy has a significant suboptimality gap, being below 90% of optimal 20% of the time, the suboptimality is not worse than the sum of that achieved by the $\theta = 0.5+$ policy in Figure 3.5.1 above, and that of the Whittle policy in [1]. We can see that the joining of the polices has not introduced significant extra error. Additionally, the median suboptimality of the best policy around 3%, which is a good performance, considering the complexity of the problem.

If we look at the worst 1% of results for the $\theta = 0.5$ with Whittle+ policy, we can see a few things. N is always at least 6, and at least one of the μ_k is high. There are no particular tendencies in the relative performance as we change K, the reward function or C. These are about what we would expect, as high N and μ_k increase the range for error and cost of an error respectively.

The jumps in the graph are because we only have a few values of K.

Our conclusion here is the same: a relatively simple policy, improved by a single step, is effective.

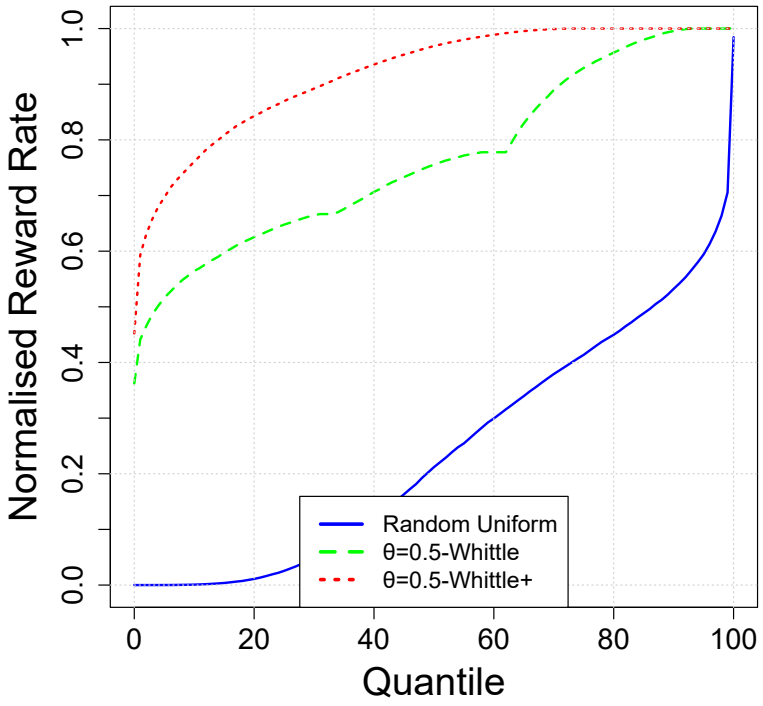


Figure 3.6.2: Policy performance for each quantile, for the full-K case.

Policy	Minimum	First Quartile	Median	Mean	Third Quartile	Maximum
Random Uniform	0.000	0.025	0.212	0.239	0.414	0.985
$\theta = 0.5$ with Whittle	0.362	0.648	0.755	0.766	0.929	1.000
$\theta = 0.5$ with Whittle+	0.452	0.870	0.969	0.920	1.000	1.000

Table 3.6.1: Mean and five-number summary for our three policies, for the full-K problem.

3.7 The Two-Phase Model

Markovian assumptions are mathematically useful, but are often inaccurate. We can take a step forwards by modelling the duration of each task as a random variable with two phases.

3.7.1 Two-Phase Formulation

Once again, we must describe a new model. The key idea is: everything remains like in Section 3.2, but each task has been split into two phases. The diagram Figure 3.7.1 may be illuminating.

We start by taking the example of the first task. We split the value x_1 into two parts x_{11} and x_{12}, representing the first and second phases respectively. If we have a failure in phase 1, then the asset moves from phase 1 to phase 2, so x_{11} decreases by one, and x_{12} increases by one. A failure in phase 2 is a 'genuine' failure, with the asset moving to the first phase of the repair process: x_{12} decreases by 1, and x_{31} increases by 1. The rates are the same for both phases, for now; see Table 3.7.1 for a complete tabulation of the transition rates.

The same applies for all tasks including the repair task, except for the reserve. As no events ever happen in the reserve, assets are only ever in the first stage, so $x_{42} = 0$ always. When we allocate assets from the reserve, they go to the first stage of the

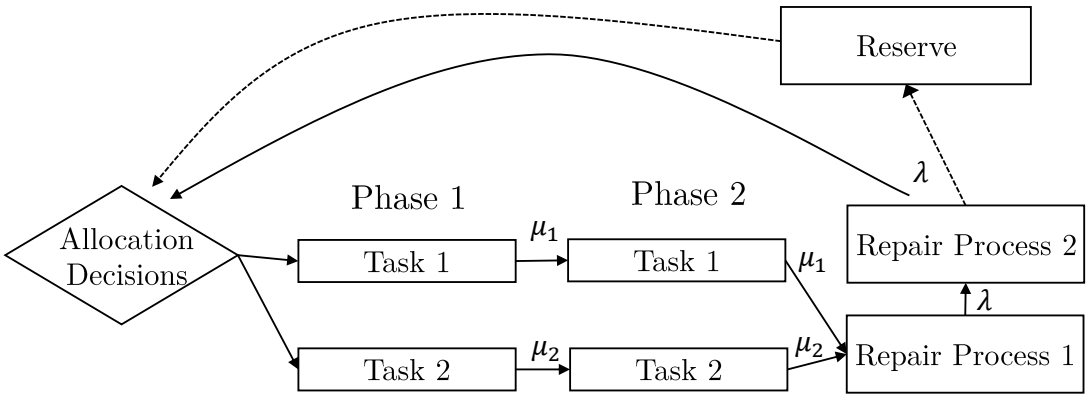


Figure 3.7.1: Flowchart of the dynamics of an asset in the two-phase case. Labels on arrows indicate rates.

relevant task.

All of the transitions - between phases and between tasks - are independent and exponentially-distributed, so that the dynamics remain Markovian. The only real change to the system is the splitting of each task into two phases.

We keep the reward functions the same as before, by simply using the number of assets in either phase for the number of assets at the task: where before we had x_1, we now have $x_{11} + x_{12}$. As such, the reward function looks like:

$$R(x) = I(x_{11} + x_{12} > C)g(x_{21} + x_{22}) \tag{3.7.1}$$

It is worth mentioning that because we have two phases, each with exponential distribution, the resultant time from the initial allocation to the asset failing has an $Erlang(2, \mu_k)$ distribution, which is also a $Gamma(2, \mu_k)$ distribution. We could nat-

Transition	Rate
$x \to x - E_{11} + E_{12}$	$\mu_1 x_{11}$
$x \to x - E_{12} + E_{31}$	$\mu_1 x_{12}$
$x \to x - E_{21} + E_{22}$	$\mu_2 x_{21}$
$x \to x - E_{22} + E_{31}$	$\mu_2 x_{22}$
$x \to x - E_{31} + E_{32}$	λx_{31}
$x \to x - E_{32} + E_{41}$	λx_{32}

Table 3.7.1: Transitions and transition rates for the two-phase case.

urally extend this to any number of phases and a $Gamma(n, \mu_k)$ distribution. The price of this additional flexibility is the size of the statespace: to treat the problem as Markovian, we have to split each task into one subtask per phase.

The use of multiple exponential phases to more accurately represent a process is common [105]. Firstly, it allows slightly more general distributions to be represented. Secondly, it well represents problems where an asset does several things in sequence. In our case, we might have a UAV travel to an area, search that area, search a second area, and then travel back. This could be represented as four different phases.

We can formulate the Bellman equation for this new scenario as follows. Write the state x as a 2×4 matrix, and define E_{ij} as the matrix with zeros in all except $E[i, j] = 1$, analagously to the standard basis vectors e_i. Then the possible non-control transitions are as as in Table 3.7.1.

The Bellman equation is hence:

$$
\begin{aligned}
&V + \phi(x) = \\
&\max_{\boldsymbol{a} \in \mathcal{A}(x)} \frac{1}{B}(R(x+a) + (B - \mu_1 x_{11} - \mu_1 x_{12} - \mu_2 x_{21} - \mu_2 x_{22} - \lambda x_{31} - \lambda x_{32})\phi(x+a) \\
&\qquad + \mu_1 x_{11}\phi(x + a - E_{11} + E_{12}) + \mu_1 x_{12}\phi(x + a - E_{12} + E_{31}) \\
&\qquad + \mu_2 x_{21}\phi(x + a - E_{21} + E_{22}) + \mu_2 x_{22}\phi(x + a - E_{22} + E_{31}) \\
&\qquad + \lambda x_{31}\phi(x + a - E_{31} + E_{32}) + \lambda x_{32}\phi(x + a - E_{32} + E_{41})) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \forall x \in \mathcal{X}
\end{aligned}
$$

Here ϕ, V, B and $\mathcal{A}$ are as before in Equation (2.2.3), and the same note applies about ineligible transitions. We are assuming that the number of assets in each phase is observable: for each asset, we know not only what task it is at, but which phase it is in.

3.7.2 Two-Phase Case Results

We run our numerical experiments across the same parameter sets as before, but with N only up to 6, as the statespace is larger for a given N. We can use the policies we already have for the one-phase problem; we do have to halve the rates in the one-phase problem to keep the problems equivalent.

We use the following policies:

- The $\theta = 0.5$ policy. This is the same regardless of the number of phases.
- The one-phase $\theta+$ policy.
- The one-phase optimal $\theta+$.
- The two-phase $\theta+$ policy.
- The two-phase optimal $\theta+$.
- The two-phase optimal policy, which is used to normalise the other policies' reward rates.

Results are depicted in Figure 3.7.2, with summary statistics in Table 3.7.2. We also display the results stratified by reward function, in Table 3.7.3.

What do we get from this? First, the results are as expected, with the two-phase optimal-$\theta+$ policy performing best. We also get the ordering of policies that we would expect.

We do see that taking into account the phases of the assets is important. The policies that use the whole two-phase structure do considerably better than those which do not, across all measures, because they have a more accurate measure of the immediate balance of failure and repair. This is nice to see: more complicated policies should give better results.

The stratified results are not particularly different; $g(x) = \min(x, 2)$ continues to be the most difficult reward function.

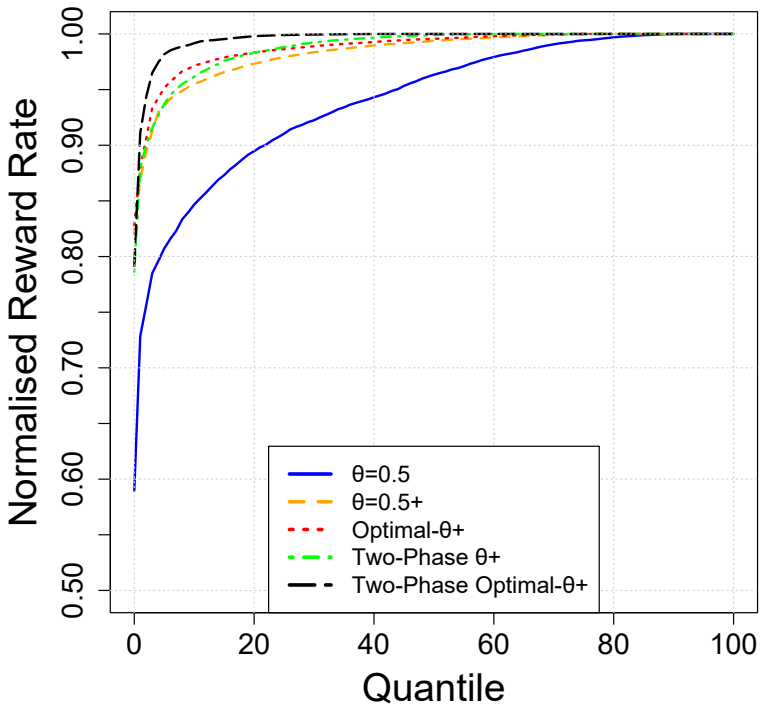


Figure 3.7.2: Policy performance for each quantile, for the two-phase problem.

Policy	Minimum	First Quartile	Median	Mean	Third Quartile	Maximum
$\theta = 0.5$	0.59	0.91	0.96	0.94	0.99	1.00
$\theta = 0.5+$	0.82	0.98	0.99	0.98	1.00	1.00
Optimal $\theta+$	0.83	0.99	1.00	0.99	1.00	1.00
Two-Phase $\theta = 0.5+$	0.79	0.99	1.00	0.99	1.00	1.00
Two-Phase Optimal $\theta+$	0.79	1.00	1.00	1.00	1.00	1.00

Table 3.7.2: Mean and five-number summary for our six policies, for the two phase problem.

Policy	Linear	Logarithmic	Minimum	Negative-Exponential	Square-Root
$\theta = 0.5$	0.85	0.96	0.87	0.93	0.98
$\theta = 0.5+$	0.96	0.99	0.97	0.98	0.99
Optimal $\theta+$	0.98	0.99	0.99	0.98	0.99
Two-Phase $\theta = 0.5+$	0.96	1.00	0.97	0.99	1.00
Two-Phase Optimal $\theta+$	1.00	1.00	0.99	1.00	1.00

Table 3.7.3: The 25^{th} quantile of the normalised reward rate, stratified by the reward function.

3.8 Non-Markovian Dynamics

An obvious flaw with the model presented in Section 3.2 above is the limitation to Markovian dynamics. While this makes both the formulation and evaluation of policies much simpler, it is inherently less realistic*. We now reformulate the problem with general distributions for both failure and repair times.

3.8.1 The Non-Markovian Model

We cannot simply replace the exponential distributions in the Markovian formulation in Section 3.2 with different probability distributions: changing the distributions introduces an element of memory to the problem, meaning we must approach the formulation from another direction, similar to how we would construct a simulation model of a non-Markovian system.

It is no longer enough to have the vector x of the number of assets at each task. Instead, we begin with a vector $\boldsymbol{y}$ of length N, where $\boldsymbol{y}_n$ is the index of the task which the n^{th} asset is allocated to at this moment in time. We do this because the formulation with x does not let us distinguish individual assets, which is essential for a non-Markovian formulation.

We also need a second vector $\boldsymbol{\tau}$ also of length N; here $\boldsymbol{\tau}_n$ is the time until something happens to the n^{th} asset. If the asset is allocated the event is a failure, if the asset is

*Unless, as mentioned before, we're studying radioactive decay.

being repaired the event is a repair, and if the asset is in the reserve then $\boldsymbol{\tau}_n$ is the time at which the policy specifies it should be allocated, should no event occur in the mean time. For any Markovian policy, $\boldsymbol{\tau}_n$ is infinite for any asset in the reserve.

We mention in passing that this asset-centered formulation is naturally suitable for heterogeneous asset problems, although we will not add that complication here.

To replace the Markovian dynamics, we require probability distributions M_k for the failure times and $L(x_{k+1})$ for the repair times; these names are deliberately reminiscent of μ_k and $\lambda(x_{K+1})$ from before. That is, when an asset is allocated to task k, we set $\tau_k \sim M_k$, and equivalently for repairs. We use the following forms for the distributions, in all cases using the same form but not necessarily parameters for repair and failure:

- Uniform distribution $U[0, 2t]$, which like the Markovian case allows failures at arbitrarily small times.
- Linearly increasing hazard-rate, with pdf $f(x) = \frac{x}{\sigma^2} e^{-x^2/(2\sigma^2)}$; this is a Rayleigh distribution, and represents failures becoming increasingly likely as assets age.
- Deterministic, with all events of the same kind taking exactly the same time t.

The Rayleigh distribution has mean $\sigma\sqrt{\pi/2}$, and if we generate a random variable $U \sim Uniform[0, 1]$ then $R = \sigma\sqrt{-2\ln(U)}$ is distributed according to a Rayleigh distribution with parameter σ. Therefore, if we generate $R = t\sqrt{2/\pi}\sqrt{-2\ln(U)}$, then the resulting random variable will be Rayleigh with mean t.

Although no Markovian-derived policy can use it, there is one extra bit of information available: the ages $\boldsymbol{a}$ of the assets at their current locations. This could be useful if we wished to take into account the likelihood of failures or repairs in the near future.

The reward structure remains unchanged, in that it is some time-invariant rate $R(x) = I(x_1 \geq C)g(x_2)$*; we limit ourselves to the two-task case for this section.

3.8.2 Policy Evaluation & Simulation

Given the above formulation, we cannot solve exactly for the average reward obtained under a policy as we did before. Instead, we must simulate the system, and evaluate the performance as such. This is unhelpful first because it takes longer, and second because it introduces a source of error previously absent: the time-averaged reward will not be exactly the long-term average. We deal with this by running the simulations for as long as proves practical, and carrying out multiple simulation runs for each policy, for each parameter set. This allows us to quantify and control the variance in the results from run-to-run; this proves to be comparatively minor compared to the means.

The simulation procedure is shown in Figure 3.8.1. The key step is that because we store the times-to-event on a per-asset basis, we can advance in time by a step of size equal to the smallest of them: by $\tau = \min_{n=1,\ldots,N} \boldsymbol{\tau}_n$.

*We could express this in terms of $\boldsymbol{y}$, but the expression becomes overly complex, and it is not difficult to convert from y to x for this calculation.

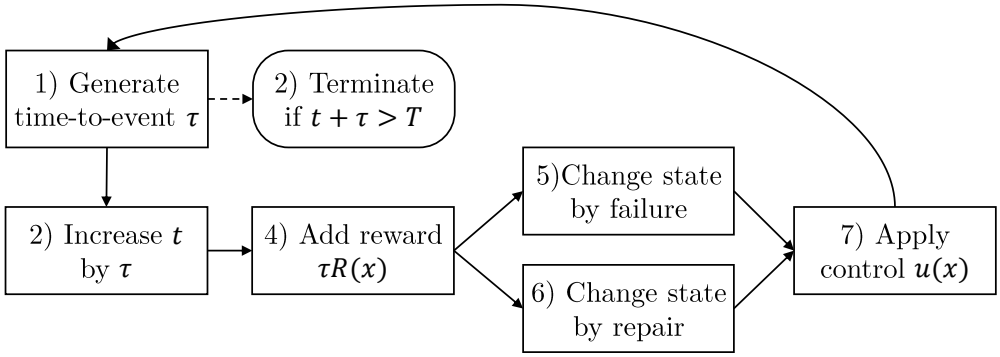


Figure 3.8.1: Simulation dynamics for the non-Markovian case.

3.8.3 Policy Translation

Our existing policies assume the problem is Markovian; a little effort is therefore necessary to apply them to the non-Markovian case. It is simple to translate between locations of assets $\boldsymbol{y}$ and numbers of assets x, but we must also provide values for the parameters $\lambda(x_3)$ and μ_k that the policies rely on.

In both cases we can use the following: if a random variable $X \sim Exp(\mu)$, then $\mathbb{E}[X] = 1/\mu$. So, if the time-to-repair for one asset being repaired is a random variable L, then we can back-define $\lambda(1) = 1/\mathbb{E}[L]$, and can do the same with the rest of the rates.

While it might seem sensible to try to instead use the remaining expected lifetime for each asset, and to generate an immediate μ or λ from that, this is not in fact possible. Firstly, the $\theta = 0.5$ and concavity policies do not actually use the rates at all. Meanwhile, the $\theta+$, optimal-θ, and optimal-$\theta+$ policies are defined and calculated

globally, and therefore we would have to recalculate the entire policy whenever we changed any of the rates.

3.8.4 Optimality and Comparison

For the model described here, it does not appear possible to find the optimal policy. While for discrete-time NMDPs it is often possible to convert them into a MDP, this cannot be done in this case, primarily because the state-space is uncountably infinite. As such, the best we can do is compare the performance of the various policies to that of the optimal policy in the Markovian case. No useful analytic upper-bound has proved forthcoming.

Even though we cannot guarantee that the Markovian-optimal policy performs best, we have still normalised so that it is at a value of 1, so that we can compare the effectiveness of the policies across different parameter settings.

3.8.5 Results and Conclusions

Our three plots Figure 3.8.2, Figure 3.8.3, and Figure 3.8.4 show the same as the graphs from earlier: the quantiles of the normalised reward rate. Note that the y-axis now included values above 1. The tables showing the summary statistics are in Table 3.8.1,Table 3.8.2, and Table 3.8.3 respectively.

We can see from these graphs that the Markovian-optimal policy is no longer always

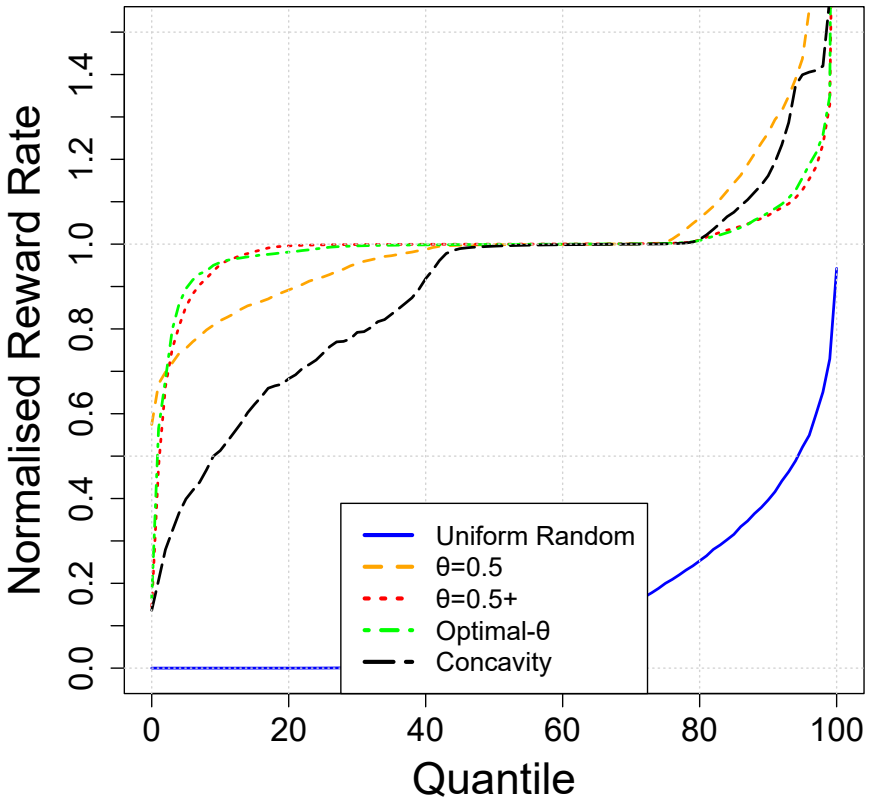


Figure 3.8.2: Simulation quantiles for the non-Markovian case with uniform random event times.

Policy	Minimum	First Quartile	Median	Mean	Third Quartile	Maximum
Random Uniform	0.0000	0.0002	0.0379	0.1275	0.2000	0.9427
$\theta = 0.5$	0.5746	0.9225	0.9986	1.0313	1.0036	5.9153
$\theta = 0.5+$	0.1449	0.9983	0.9999	0.9978	1.0018	2.5598
Optimal θ	0.1654	0.9908	0.9993	1.0014	1.0016	3.7564
Concavity	0.1363	0.7423	0.9952	0.9027	1.0006	3.3275

Table 3.8.1: Mean and five-number summary for our five policies, for the non-Markovian case with uniform random event times.

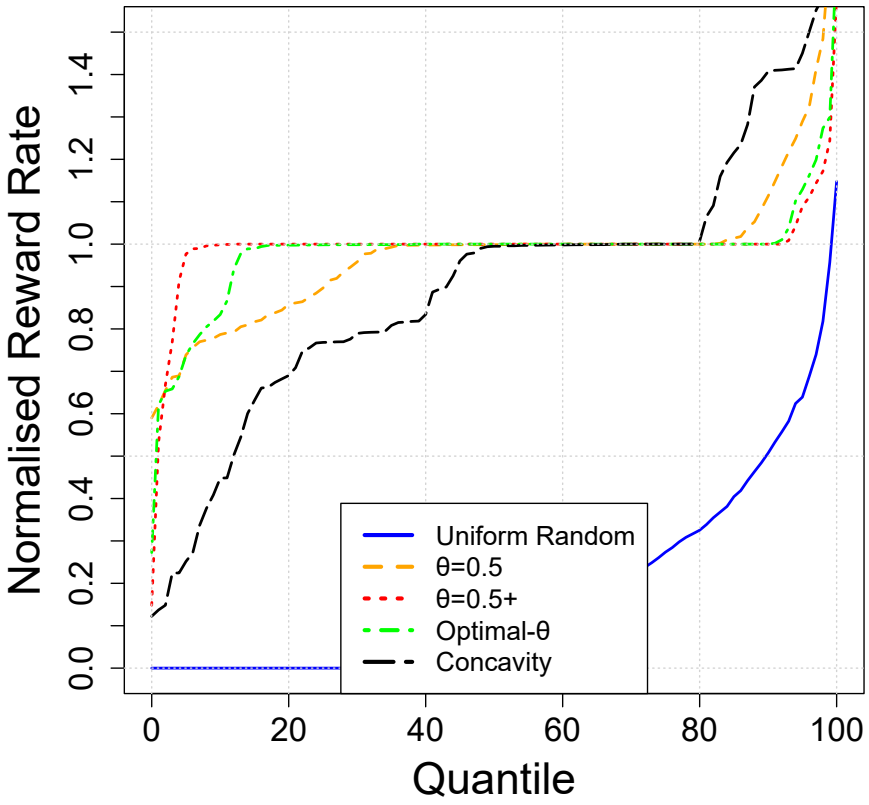


Figure 3.8.3: Simulation quantiles for the non-Markovian case with deterministic event times.

Policy	Minimum	First Quartile	Median	Mean	Third Quartile	Maximum
Random Uniform	0.000	0.000	0.022	0.163	0.274	1.148
$\theta = 0.5$	0.590	0.896	0.999	0.982	1.000	3.670
$\theta = 0.5+$	0.148	1.000	1.000	0.996	1.000	1.585
Optimal θ	0.273	0.999	1.000	0.983	1.000	1.693
Concavity	0.122	0.768	0.995	0.908	1.000	1.989

Table 3.8.2: Mean and five-number summary for our five policies, for the non-Markovian case with deterministic event times.

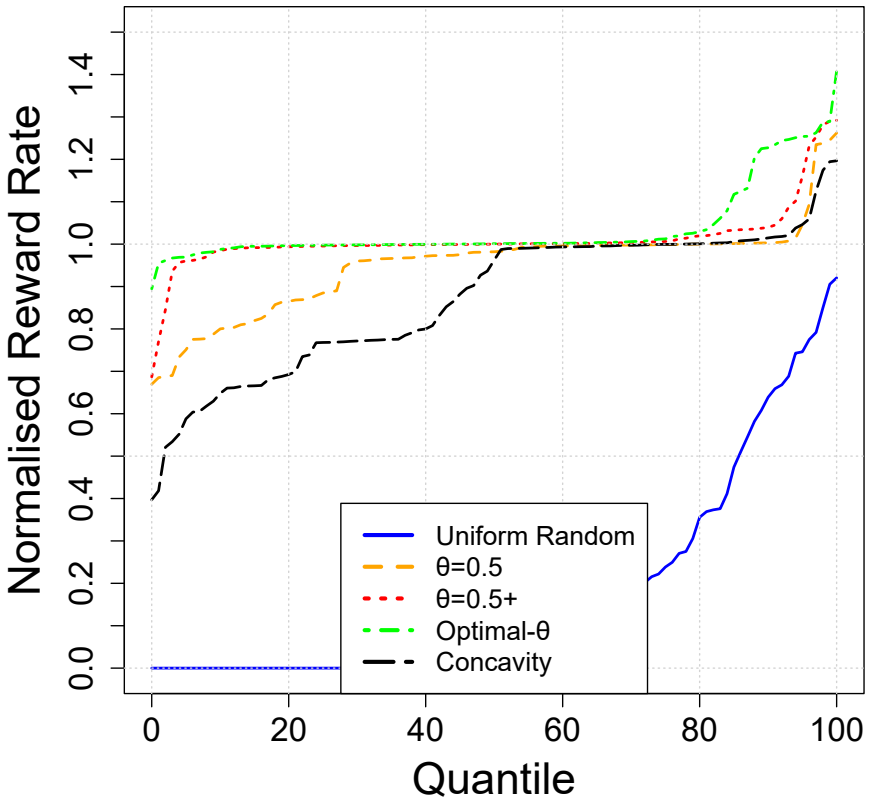


Figure 3.8.4: Simulation quantiles for the non-Markovian case with Rayleigh event times.

Policy	Minimum	First Quartile	Median	Mean	Third Quartile	Maximum
Random Uniform	0.000	0.000	0.012	0.168	0.238	0.920
$\theta = 0.5$	0.669	0.885	0.982	0.949	0.999	1.262
$\theta = 0.5+$	0.687	0.996	1.000	1.011	1.006	1.292
Optimal θ	0.894	0.998	1.001	1.036	1.012	1.409
Concavity	0.397	0.768	0.962	0.871	0.999	1.197

Table 3.8.3: Mean and five-number summary for our five policies, for the non-Markovian case with Rayleigh event times.

the best, and that the $\theta+$ policy remains reliable. It is impressive how robust the $\theta+$ policy is to changes in the precise structure of the problem.

We can also see that some policies are 'safer' than others. In particular, the concavity policy is worse than the Markovian-optimal in about 40% of cases, but is better in about 20%. Meanwhile, the optimal-θ policy deviates below and above in about 15% and 20% of cases respectively.

Finally, there does not seem to be any significant differences between the relative performances of the policies with respect to the precise distributions used.

3.9 Future Work

First: we can conclude that the optimal-$\theta+$ policy is effective, robust, and computationally practicable.

Be that as it may, there are several possible complications that could be added:

1) Adding travel costs or delays of some form. This would not in itself be particularly difficult: to add travel costs to the model is easy, and travel times can be represented by travel costs, if a little indirectly. First, travelling to and from the repair task can be incorporated into the time taken for the repairs. Now suppose it takes a time τ_k to send an asset from the reserve to task k. If the travel times are small enough that sent assets are likely to arrive before the next failure or repair, then instead impose a charge $\tau_k(R(x + e_k - e_4) - R(x))$. That is, we lose the difference in rewards for a

length of time τ_k.

Without any particular reason to add this to the model, we are inclined to leave such matters out.

2) Heterogeneous assets. As we mentioned in Section 3.8.1, the formulation using the vector $\boldsymbol{y}$ instead of x allows us to easily introduce heterogeneity into the model, because we are then keeping track of every asset's position. We mentioned in Section 1.1.1 that this would be common in S&R problems, so this strikes us as an addition worth future investigation.

3) Excessively large N, for example more than 10^4. Unless some application with such large N presents itself, does not seem useful enough to justify further investigation. It is worth mentioning that all of the numerical experiments above were done with $N \leq 10$, while we would expect our real-world problems to have $10 \leq N \leq 100$. This size was chosen purely so that we could compute the optimal policy, so that we had something to compare our heuristics to.

4) Imperfect observation of states*, leading to a POMDP [106]. This sort of formulation is unlikely to be suitable for our problem: we should know where the assets are and what they are doing.

5) Initially unknown rewards and communications limits. The framework here is analogous to a bandit problem [28]: each time we reach a state x, we would get reward as a Poisson process with some rate $r(\boldsymbol{x})$, until the state changes. We would

*These could be x or y.

have to learn about the rewards gained per state (and need to specify priors for them), as we try to stay in high-reward states. Unlike a bandit problem, we have a complicated state-space, and the dynamics are not entirely under our control.

This problem, considered quite generically, is really a disguised form of reinforcement learning [107]. As such, existing methods could be brought to bear on this case.

Chapter 4: Conclusions

In this chapter, we first recapitulate what we have found, and the lessons that we have learned from this thesis. We then talk about how the results in this thesis can be put to practical use, before discussing how the results could be built upon.

4.1 Findings

In the previous two chapters we have found policies close to optimal in performance but still computationally tractable. The second facet is worth stressing: for the independent case the clever Whittle policy is practically instantaneous compared to full policy improvement, while for the dependent case look again at Figure 3.5.2 at the relative speeds of the heuristics. This would only be amplified if we increased the number of assets N to the 10-100 likely in real-world applications. For the dependent case we also have a nice series of policies which increase in effectiveness and computational cost together, so that if we need a quick answer, we can obtain one at the cost of worse performance.

For the independent case in Chapter 2, the policy we started with was a Whittle index policy [31], arising by considering the problem as a restless bandit, and using that to find a 'price' that each task would be willing to pay for an asset. This general idea of pricing as a control approach is a very 'economics' approach to the problem, and suitable for generalisation to similar problems, where the tasks all work the same way but with different parameters. We discuss this more in Section 4.4

For the dependent case in Chapter 3, we had to be more inventive. We started by creating what we have called the policy framework: a structure that limits the policies that we have to consider in such a way that we do not eliminate the optimal policy*. This then naturally gives us a simple way to generate a whole family of policies - the θ-policies - which we have every reason to believe have performance close to optimal.

As these policies are parameterised by a single parameter - θ - we can quite easily find the optimal θ, in the sense of the one that gives the best time-averaged infinite-horizon reward. We can then apply a single step of policy improvement to the resultant policy, as before. This is both welcome and reasonable: we encode our problem-specific knowledge into a policy, then improve it by an extra step of foresight.

Additionally, for the dependent case, we have shown that this form of policy is robust to the precise formulation, and performs well regardless. This included going from purely exponential distributions to a two-phase Erlang formulation, which, while not the greatest of deviations, still allows us more room to fit to real-world distributions.

*The optimal policy is guaranteed to fit into the policy framework. It is not guaranteed to be a θ-policy.

4.2 Lessons Learned

First, we have learned about the inherent ease-of-use of Markov decision processes, at least compared to non-Markovian ones. Not only do they have a nice clear structure which makes formulation easier, it is as simple to evaluate policies* as anything involving linear algebra ever could be. A Markovian formulation also allows us to use policy improvement, and lets us find a definitively optimal policy, at least for small N. This lets us compare any policies that we formulate to the genuine optimal policy, giving a definite answer to how well they are performing.

As mentioned above, this only works for N smaller than is of practical use; were this not the case we would have no need for heuristics.

Second, coding up mathematical algorithms is often much harder than it should be, simply because mathematicans and computers naturally think of things in different ways. In this thesis this showed up most strongly in producing the mappings between statespaces and numbers $1, \ldots, N$, especially because it is a detail almost always glossed over in textbooks and courses. The policy improvement code was also a significant time sink, and took multiple restarts to get to work - even simple things like generating all $a \in \mathcal{A}(x)$ turn out to be hard to program.

Third, it is better, in the sense of being more productive, to sort out a limited case well than to flail about on a more general or more applicable case. If you cannot

*As in, it takes $O(|\mathcal{X}|^3)$ time.

produce anything useful, it is irrelevant how good the model was or how closely it sticks to reality - the point of modelling is to provide solutions, not to look good.

We can see this in our foray into a non-Markovian model. While this is certainly more realistic than just assuming that all of the relevant distributions are exponential, we cannot find the optimal policy or even analytic lower or upper bounds. While we can at least evaluate policies through simulation, we can only compare them to each other, which does not tell us if their performance is actually close to the best possible.

Finally, it is not a bad idea to take multiple approaches to a problem, whether that means multiple policies or just several different formulations. Even a failed approach is a learning experience, and finding that something definitely doesn't work can be as productive as a more obvious success.

4.3 Applying These Results

We have, in the previous chapter, been more concerned with formulating and optimising than with actually applying the results; this is unfortunate, but inevitable. We now discuss how the contents of this thesis might be bent towards some practical good.

The first point is this: know your problem*. Even given a formulation, there are a huge number of parameters that need to be extracted from reality and piped into the

*ΓΝΩΘΙ ΣΑΥΤΟΝ

model: failure rates, repair rates, estimated reward rates, and more.

It is therefore crucial to understand what you are asking for, and what you can provide. Our work shows that the optimal policy varies significantly depending on the parameters, and therefore that simply making them up is insufficient. Similarly, do you need a complete policy, one that specifies your decisions under all circumstances? Or do you just want guidance and advice?

If what you want is the former, then you need to think about adopting the modelling in this thesis wholesale. The latter strikes me as more sensible; an example would be the use of θ-policies as a guide. That being said, using a model to guide your thinking gives three possibilities:

1. The model suggests what you would have done anyway. Then you can go ahead with what you would have done anyway.

2. The model suggests allocations slightly different from what you would have done anyway. Try what the model suggests, but monitor its performance, and be ready to revert.

3. The model suggests an entirely different set of allocations from what you would have done anyway. **Stop and think!** This is a sign that in this case, either you or the model is missing something. Think about which it might be. Consider the possible consequences of following the model's allocations; how bad could it get? Follow the model only if you have spotted something you've missed, or if the probable consequences are minor. Take care, have a backup plan, and pay

close attention to what happens.

It must be said that the three points above apply to any set of circumstances where you are applying a model-based approach to situations where people have been taking actions for some time without any mathematical guidance whatsoever.

The second point is: look one step ahead. This is the essence of policy improvement[24], which uses an existing policy to give a relative value to states, and then tries to maximise the sum of reward we'll get at this state, and expected value of the state we transition to next.

This does not have to be understood merely mathematically. You can think in terms of relative value of states, even if your measure of that value is quite crude. Given this, you can utilise the RHS of the Bellman equation:

$$\max_{a \in \mathcal{A}(x)} (R(x,a) + \sum_{y \in \mathcal{X}} P(x,y|a)\phi(y)) \tag{4.3.1}$$

The key part remaining is: what about $a \in \mathcal{A}(x)$? Again, this is where subject expertise comes into play. Ask yourself: what can you do? What can't you do? Why not? What would it cost you to do that? What would you have to change to be able to do that?

The procedure is then:

1. What is $\mathcal{A}(x)$? What allocations could you apply, in the current moment?

2. How does $R(x, a)$ work? How does your action affect the reward you will gain in the immediate future?

3. What are the important $P(x, y|a)$? How does your action affect the system? What kind of failures or repairs are likely?

4. What are the $\phi(y)$? How much would you prefer to be in one state instead of another? Can you rank the possible states? Which states do you want to avoid? Which states do you want to be in?

Lastly: build a policy framework. We did this in Chapter 3, identifying features than an optimal policy would have to have, and zeroing in on the remaining uncertainties. Similarly, you can try to rule out things you know you would never want to do, and try to concentrate on the genuinely hard decisions.

To summarise what we have learnt about deploying assets with failure and replenishment: know your problem, look one* step ahead, and minimise the number of choices you have to make.

4.4 Intensifications

We now talk about two related topics: the first is where we could have gone into more detail on things we have covered in this thesis, while the second is things that we have

*Or more if possible, but the point is that looking even one step ahead is a major improvement over not looking ahead at all.

not properly covered that it would be interesting or useful to attack. This section addresses the first.

The inevitable question about this subsection is: why did we not do these things? The primary reason for failing to address these complications is simple lack of time - there were only so many things that we could cover. Part of this was that some matters - the non-Markovian case in particular - were particularly difficult and hence would have required an undue amount of extra time and attention.

A secondary difficulty was lack of knowledge. While I do possess a passing familiarity with such topics as parallellisation, reinforcement learning and approximate dynamic programming, I did not start with any great expertise in them, and so my attention was naturally directed to areas which I had a more thorough grounding in. Of course this links to the previous difficulty, as such things are not impossible to learn, but it seemed more productive to work with what matters I could.

A third reason is that certain complications, whilst natural extensions, did not seem particularly interesting. Increasing N, using a different programming language, considering heterogeneous assets - none of these seemed to justify the time that would have had to be spent on them.

One obvious but not necessarily interesting extra is simply increasing the size of the problem. Using the code I have produced, anything above about $N = 15$ takes an irritatingly long amount of time to calculate the optimal policy for*, which is why we

*Especially because we were running thousands of cases.

did not do so. But there are certain changes that could be made to ameliorate this, the first of which is changing to sparse matrices. Transition matrices are often sparse [108], and this would greatly reduce the memory usage.

Similarly, when testing the policies in different scenarios, it would be possible to run several scenarios at a time, in parallel. This is not easy, and we did make attempts at this which only served to cause complicated and inexplicable errors that were judged not worth the time to resolve.

R as a programming language is also not the quickest; it was chosen for its ease of use, familiarity, and because it allows for rapid prototyping. It also has the advantage of generally readable code. If speed was the primary concern, a language such as C++, which has plenty of linear algebra libraries, would be preferable.

Similarly, Chapter 3 was limited in the forms that it looked at, and we could certainly look at dependence structures inspired by different problems, such as S&R or project management. One worth looking at is a generalisation of the one we had, where for each number of assets communicating x_1, a certain number of assets $f(x_1)$ can communicate back; f should start at 0 and be strictly increasing. The reward function would therefore become:

$$R(x) = g(\min(f(x_1), x_2)) \tag{4.4.1}$$

The previous case then corresponds to $f(x) = 0$ for $x < C$ and $f(x) = N$ for $x \geq C$.

This case is more general but still nicely limited, and I suspect would prove amenable to a slight generalisation of the policy framework.

To go into more detail on this, we still have the same simplification where if $f(x_1) > x_2$ there is no point sending more assets to task 1. But now when considering an allocation, in addition to the previous case when $f(x_1) > x_2$ and our only choice is reserve vs. allocate, we also have the case where $f(x_1) = x_2$ and we need to decide if it is worth sending one more asset to task 1 in order to be allowed to send up to $f(x_1 + 1) - f(x_1)$ assets to task 2.

We could add another parameter σ; as θ controls the balance between allocation to task 2 and reserving, σ would control the balance to allocating to task 1 or task 2 and reserving. This is only a sketch; much more work would be needed to turn this into a coherent policy.

There are other possible dependence structures:

- $R(x) = g(\min(x_1, C, x_2))$, linear up to a threshold; suitable for where we have restrictions on both commications sending (x_1) and receiveing (C).
- $R(x) = \max(g_1(x_1), \ldots, g_K(x_k))$, where we get the best of what we have; suitable for when several tasks are independently searching for the same target.
- $R(x) = \min(g_1(x_1), \ldots, g_K(x_k))$, where we are rewarded according to the weakest link; suitable for a facility defense or perimeter patrol problem.
- $R(x) = \prod_{k=1}^{k=K} g_k(x_k)$, where we are rewarded for keeping all tasks filled at once;

could also be used for defense problems, but encourages us to fill up all tasks, not just the most important.

Two other choices we made in the formulations were to look at the time-averaged infinite-horizon rewards, and to have all assets identical. Discounted rewards would not be significantly different, as the Bellman equation is still applicable [24], with the addition of a single occurence of the discount factor β, and the removal of the average reward*:

$$\phi(x) = \max_{a \in \mathcal{A}(x)} (R(x,a) + \beta \sum_{y \in \mathcal{X}} P(x,y|a)\phi(y)) \tag{4.4.2}$$

This would probably not change the effectiveness of our policies, but that is the whole point: until we look, we cannot be sure. There is no reason that changing to discounted rewards would affect the computational time taken, so the same one step of policy improvement approach should remain computationally feasible

Finite horizons are a little trickier, as we have to take into account the time until the horizon, but this does not make things fundamentally different. We can still use the Bellman equation, but we work by reverse induction, using the time-to-go as our variable: at the horizon, we can get no more rewards, so we know what happens at the edge, and can work backwards from that. This is all standard stuff; the point is that this model could be applied to different cases, and it would be interesting to see

*Because the average reward is zero, in the long run, as long as the rewards at any step are bounded in absolute value.

if the same policies still worked well.

The use of heterogeneous assets, as often arise in S&R problems (see Section 1.1.1), however, works somewhat differently. It would probably be better to use the formulation given in the non-Markovian section of Chapter 3, with the state being described by a vector $\boldsymbol{y}$ of the locations of the assets. But even if the dynamics are still Markovian, this makes the state-space much larger. Be that as it may, heterogeneous assets are mentioned often in the literature [109, 110, 111] and so allowing for this would be useful, to say the least.

4.5 Extensions

Let us now go from intensifications to extensions.

Learning the Rewards

The most potentially productive extension of this work which I have identified is that of initially unknown rewards. In reality we would reasonably expect to have a good idea of some parameters, such as N, λ and the μ_k, but the reward functions - $R(x)$ - are quite possibly entirely unknown. Bandit problems are all about the balance of exploration and exploitation [112, 113, 114], and the whole point of searching is that we want to find something, but we do not know where it is. Similarly, in environmental modelling we might not know which places are important to monitor, and in project

management we might not know how productive individuals will be.

So suppose the reward functions are initally unknown, but at least fixed in time*. What can we do?

First, the idea of 'learning' only makes sense in either finite or discounted time [107]. If we measure performance by the time-averaged infinite-horizon reward, we could spend an arbitrary amount of time learning about the system and our performance would not be adversely affected. Therefore we need to use either finite-horizon or discounted rewards.

For various reasons, most notably the removal of boundary effects, I would go with discounted rewards, which also have a nice practical justification. Suppose that, in discrete time, the system as a whole ends with some probability $1-\beta$ at each timestep. Therefore, we only make it to timestep t with probability β^t, and so we must discount rewards accordingly. Similarly for continuous time, we have the system-ending event happening at a constant rate β, and so rewards get discounted according to $e^{-\beta t}$.

If the reward functions are unknown but deterministic, we would learn about them instantly - as soon as we first reach a state x, we would learn the exact value of $R(x)$. This is unrealistic and far too easy, and it is therefore necessary to make the process of gaining rewards to be in some way random. The obvious way to do this is to have binary rewards arrive (in some sense) at a rate $R(x)$. In discrete time that would mean that when in a state x, we would get rewards at each step according to

*Learning about things that change in time would be an entire Ph.D. on its own.

a distribution $Bernoulli(R(x))$, with $R(x) \in [0, 1]$.

For a uniformised continuous-time version, we would have rewards, each of value 1, arriving according to a Poisson process of rate $R(x)$, giving rise to a total number of rewards accrued between jumps* which is distributed according to a geometric distribution; for a derivation, see Appendix D.

Unfortunately, this breaks things, as the variance is excessive compared to the mean; if a random variable $X \sim Geo(p)$, then it has mean $1/p$ and variance $\frac{1}{p^2} - \frac{1}{p}$. For our problems, p is probably low, making the variance much higher than the mean. This suggests that either we should move to a discrete-time framework, or make the transitions deterministic. The latter option is not as unreasonable as it sounds - we would essentially be looking at this like a bandit problem, just with constrained controls: our control would be the vector x, constrained by what it was at the previous time-step.

We might also have to learn the μ_k, but this is less complicated, being just a statistical estimation problem. But it is yet another difficulty to add.

NMDPs

The second major extension would be to actually get somewhere with the non-Markovian case. This is of course easier said than done, but important if we want to accurately model real-world behaviour - the life-spans of deployed drones are unlikely

*Recall that in a uniformised chain jumps happen at a constant rate B.

to be accurately modelled by exponential distributions. I do not know how to do this; if I did, I would have done so already. Additionally, most examples of NMDPs in the literature [115, 116] deal with them by transforming them into MDPs.

The key difficulty, that we mentioned in section 3.8, is that we cannot find the optimal policy. There are three obvious approaches that we could take towards this.

The first approach is what we did: don't do anything. Rely on comparisons between policies, and on our knowledge of what sensible policies should do. While this has the advantage of simplicity, this does not really work because we have many scenarios that we want policies to work well on, and different policies do better on different scenarios.

The second approach is to find upper and lower bounds on the reward rates, and compare our policies to them. This is not as effective, but should be easier.

The third approach is to change the details of the problem until we can actually find the optimal policy. This seems sensible, but the reason we wanted to switch to a NMDP was to make our model more realistic; having to then simplify the model rather undermines the point.

Approximate Dynamic Programming

We mentioned the effects of increasing N and K above, but particularly if we want to include multiple phases in our model as in Section 3.7, the statespace increases in size

dramatically; this would be particualrly important for the naval search applications, where we might have a large number of assets (N high) and a large number of different places to search (K high). If we have N assets and M places* to put them, then we have:

$$|\mathcal{X}| = \left(\begin{array}{c} N+M-1 \\ \\ M-1 \end{array} \right) = \frac{(N+M-1)!}{(M-1)!N!} \tag{4.5.1}$$

This is roughly $N^M/M!$, and we would expect $N \gg M$ for our applications. So we have a statespace that quickly gets too large even to enumerate, let alone perform operations on. If the assets are heterogeneous and all distinct matters are even worse, as we then have exactly N^M possible states.

So what are we to do? While there are always tricks that can be employed to shrink the effective statespace, what we are going to talk about now is approaches that let us only handle the parts of the statespace we need. This is called approximate dynamic programming (ADP) [117]. In ADP, just like in standard dynamic programming, we have states x, controls a, transition probabilities $P(x, y|a)$, and a reward function $R(x, a)$, although these often have different names†. We use again the Bellman equation, which we present in the discounted form:

$$\phi(x) = \max_{a \in \mathcal{A}(x)} (R(x, a) + \beta \sum_{y \in \mathcal{X}} P(x, y|a)\phi(y)) \tag{4.5.2}$$

*We would have to have one for each combination of phase and task.

†This is because there are several different fields that use dynamic programming techniques.

To quote from a good overview of the topic [118]:

"The essence of approximate dynamic programming is to replace the true value function with some sort of statistical approximation."

All of a sudden, we have gone from $|\mathcal{X}| = N^M$ to something much smaller, which we can actually work with. Rather than give all the details, we refer you again to [118]; the more important question is how this would work for us.

One common idea in ADP is taking sample paths of the future, and updating ϕ based on this. This is very well suited for our problem, where there are many states we should never get into - all assets in the reserve, for example. Any use of a policy framework as in Chapter 3 also restricts the size of the portion of the state-space that we actually ever visit, hopefully to something computationally practicable.

Generalised Bandits

What if we tried to formulate our problem more like a bandit problem? This would be most appropriate for the S&R-inspired problems, but, if it works, could be applied elsewhere. For this we limit ourself to the independent case, as bandits are required to be independent of each other in both dynamics and rewards. We have bandits $1, \ldots, K$, which are our tasks. Each gives binary rewards out randomly at a rate $g_k(x_k)$ when we have x_k assets at that bandit.

Now for the key switch: x is our control, not our state. The state is the observed

rewards gained, per task when we had a specific number of assets there. In the simplest case, at each discrete timestep $t = 0, 1, \ldots$ we choose a vector x such that $\sum_{k=1}^{K} x_k = N$. This would give us a bandit with bunched-delayed rewards, which is somewhat non-standard, but has been studied [119, 120, 121].

But that is not our problem. The whole point of our problem is that we have constraints on how the system-control goes! Incorporating this would be tricky, and we can only present some thoughts on how. One option is to impose probabilistic contraints on the control. Essentially, this would be a Lagrangian relaxation of the problem, so that under our policy U:

$$\sum_{k=1}^{K} p(x_k(t+1) = i+1 | x_k(t) = i) = \frac{\lambda}{B + \lambda + \sum x_k(t)\mu_k} \tag{4.5.3}$$

$$\mathbb{E}_t[p(x_k(t+1) = i-1 | x_k(t) = i] = \frac{\mu_k x_k(t)}{B + \lambda + \sum x_k(t)\mu_k} \ \forall k \tag{4.5.4}$$

The first condition says that the average arrival rate of assets across all tasks is λ, while the second says that the expected departure rate of assets from task k is $\mu_k x_k(t)$. The constant B is the uniformisation constant, and $\frac{B}{B+\lambda+\sum x_k(t)\mu_k}$ is the probability the state does not change.

This setup does not include an explicit reserve, but that is another reason for the Lagrangian relaxation: we can take assets from one time and use them at another, which is the essence of the reserve.

So that is a Markovian pseudo-bandit setup for our problem. The reason for doing

this is that we can try and apply standard bandit techniques [28][122], and not have to invent approaches entirely from scratch. This is merely a sketch of a possible approach, but one that strikes me as plausible and worth investigating.

Appendix A: Proving The Identity

This appendix proves the identity 2.3.19 used in 2.3.2.

We wish to prove that:

$$w(x) = \frac{\sum_{y=0}^{x} P_y(g(y+1) - g(y))}{\mu \sum_{y=0}^{x} P_y} \tag{A.0.1}$$

Begin with the expression for $w(x)$:

$$w(x) = \frac{\sum_{y=0}^{x+1} g(y)\pi_{x+1}(y) - \sum_{y=0}^{x} g(y)\pi_x(y)}{\Lambda(\pi_x(x) - \pi_{x+1}(x+1))} \tag{A.0.2}$$

Next we define

$$\rho = \frac{\Lambda}{\mu} \tag{A.0.3}$$

Recall from equation (2.3.18) that $P_x = \sum_{y=0}^{x} \rho^y / y!$, which is the normalising constant for $\pi_x(x)$. The second sum in the numerator of $w(x)$ can be written as:

$$\sum_{y=0}^{x} \pi_x(y)g(y) = \frac{1}{P_x}\left(\sum_{y=0}^{x} \frac{\rho^y}{y!}g(y)\right) \tag{A.0.4}$$

Take out a term $g(x)$ from the sum:

$$\sum_{y=0}^{x} \pi_x(y)g(y) = \frac{1}{P_x}\left(\left(\sum_{y=0}^{x} \frac{\rho^y}{y!}\right)g(x) + \sum_{y=0}^{x-1} \frac{\rho^y}{y!}(g(y) - g(x))\right) \tag{A.0.5}$$

Using the definition of P_x:

$$\sum_{y=0}^{x} \pi_x(y)g(y) = \frac{1}{P_x}\left(P_x g(x) + \sum_{y=0}^{x-1} \frac{\rho^y}{y!}(g(y) - g(x))\right) \tag{A.0.6}$$

We can repeat what we just did:

$$\sum_{y=0}^{x} \pi_x(y)g(y) = \frac{1}{P_x}\left(P_x g(x) + \sum_{y=0}^{x-1} \frac{\rho^y}{y!}(g(x-1) - g(x)) + \sum_{y=0}^{x-1} \frac{\rho^y}{y!}(g(y) - g(x-1))\right) \tag{A.0.7}$$

Rewrite, noting that the last term of the second sum is zero:

$$= \frac{1}{P_x}\left(P_x g(x) + P_{x-1}(g(x-1) - g(x)) + \sum_{y=0}^{x-2} \frac{\rho^y}{y!}(g(y) - g(x-1))\right) \tag{A.0.8}$$

By doing this repeatedly, we can collapse the whole sum:

$$\sum_{y=0}^{x} \pi_x(y)g(y) = \frac{1}{P_x}\left(g(x)P_x + \sum_{y=0}^{x-1} P_y(g(y) - g(y+1))\right) \tag{A.0.9}$$

Now look at the numerator of the formula for $w(x)$. Using the expression just obtained, we can express it as:

$$\frac{1}{P_{x+1}}\left(g(x+1)P_{x+1} + \sum_{y=0}^{x} P_y(g(y) - g(y+1))\right) - \frac{1}{P_x}\left(g(x)P_x + \sum_{y=0}^{x-1} P_y(g(y) - g(y+1))\right) \tag{A.0.10}$$

Joining the two sums together:

$$= \frac{P_x}{P_x}(g(x+1) - g(x)) + \frac{P_x(g(x) - g(x+1)}{P_{x+1}} + (\frac{1}{P_{x+1}} - \frac{1}{P_x})\sum_{y=0}^{x-1} P_y(g(y) - g(y+1)) \tag{A.0.11}$$

The numerator is therefore equal to:

$$\frac{P_{x+1} - P_x}{P_{x+1}P_x}\left(\sum_{y=0}^{x} P_y(g(y+1) - g(y))\right) \tag{A.0.12}$$

For the denominator of $w(x)$, we can simplify directly:

$$\pi_x(x) - \pi_{x+1}(x+1) = \frac{P_x - P_{x-1}}{P_x} - \frac{P_{x+1} - P_x}{P_{x+1}} = \frac{P_x^2 - P_{x+1}P_{x-1}}{P_{x+1}P_x} \tag{A.0.13}$$

We can now combine the two expressions we have obtained to get a new form for $w(x)$, with a factor of $P_x P_{x+1}$ cancelling:

$$w(x) = \frac{1}{\Lambda} \frac{(P_{x+1} - P_x)\left(\sum_{y=0}^{x} P_y(g(y+1) - g(y))\right)}{P_x^2 - P_{x+1}P_{x-1}} \tag{A.0.14}$$

We can simplify the terms involving P_x by rearranging as follows:

$$\frac{(P_{x+1} - P_x)}{{P_x}^2 - P_{x+1}P_{x-1}} = \frac{\rho^{x+1}/(x+1)!}{P_x^2 - (P_x - \frac{\rho^x}{x!})(P_x + \frac{\rho^{x+1}}{(x+1)!})} \tag{A.0.15}$$

Multiply out the brackets on the bottom and cancel:

$$= \frac{\rho^{x+1}/(x+1)!}{P_x(\frac{\rho^x}{x!} - \frac{\rho^{x+1}}{(x+1)!}) + \frac{\rho^x}{x!}\frac{\rho^{x+1}}{(x+1)!}} \tag{A.0.16}$$

Strip out a factor of $\rho^{x+1}/(x+1)!$ from top and bottom:

$$= \frac{1}{P_x(\frac{x+1}{\rho} - 1) + \frac{\rho^x}{x!}} = \frac{1}{P_x\frac{x+1}{\rho} - P_x + \frac{\rho^x}{x!}} = \frac{1}{P_x\frac{x+1}{\rho} - P_{x-1}} \tag{A.0.17}$$

Simplifying this expression requires us to prove a lemma.

Lemma: $P_x\frac{x+1}{\rho} - P_{x-1} = \frac{1}{\rho}\sum_{y=0}^{x} P_y$

Proof: We proceed by induction. For $x = 1$ we substitute in $P_0 = 1$ and $P_1 = 1 + \rho$:

$$(1+\rho)\frac{2}{\rho} - 1 = \frac{1}{\rho}(1+1+\rho) \tag{A.0.18}$$

$$\frac{2}{\rho} + 2 - 1 = \frac{1}{\rho} + \frac{1}{\rho} + 1 \tag{A.0.19}$$

The two sides are equal, so we have our base case.

For the induction step, we assume the $x-1$ case, that:

$$P_{x-1}\frac{x}{\rho} - P_{x-2} = \frac{1}{\rho}\sum_{y=0}^{x-1} P_y \tag{A.0.20}$$

When we go to the x case, we must add the same to both sides; we must have that:

$$P_x\frac{x+1}{\rho} - P_{x-1}\frac{x}{\rho} - \frac{\rho^{x-1}}{(x-1)!} = \frac{1}{\rho}P_x \tag{A.0.21}$$

We can manipulate the L.H.S as follows, beginning by taking out a factor of ρ^{-1}:

$$\frac{1}{\rho}\left((P_{x-1} + \frac{\rho^x}{x!})(x+1) - P_{x-1}x - \frac{\rho^x}{(x-1)!}\right) = \frac{1}{\rho}P_x \tag{A.0.22}$$

Cancel the factor ρ^{-1}, and multiply out the brackets:

$$xP_{x-1} + P_{x-1} + \frac{x\rho^x}{x!} + \frac{\rho^x}{x!} - P_{x-1}x - \frac{\rho^x}{(x-1)!} = P_x \tag{A.0.23}$$

$$P_{x-1} + \frac{\rho^x}{x!} + \frac{\rho^x}{(x-1)!} - \frac{\rho^x}{(x-1)!} = P_x \tag{A.0.24}$$

$$P_{x-1} + \frac{\rho^x}{x!} = P_x \tag{A.0.25}$$

We have hence established the required identity $\square$.

We can now use this result and rewrite w_x as:

$$w(x) = \frac{1}{\mu} \frac{\sum_{y=0}^{x} P_y(g(y+1) - g(y)}{\sum_{y=0}^{x} P_y} \tag{A.0.26}$$

This is the required identity $\square$.

Appendix B: Algorithm Pseudocode

This appendix contains pseudocode of the significant algorithms used in this thesis, emphasising where matters are different from the normal. These are more detailed than pseudocode normally is, as the implementation-specific details are often the most interesting part.

Throughout, $\mathbf{0}$ is the vector of all zeros, and e_i is the i^{th} basis vector, both in the appropriate number of dimensions. Square brackets show access to an element of an array, while round brackets show a call to a function. = is used for assignment, which == is used for checking equality. Comments are preceded by two hashes and are in italics ## *like so*. Syntax is derived from a combination of R as it is what these algorithms were coded in, and from the defaults for the algorithmicx package used to typeset the algorithms.

B.1 Policy Improvement

This is the algorithm for policy improvement as described in Section 1.2.1. Two things are unusual: this one actually explains the workings in detail, and the use of impulsive controls.

U is the policy, represented as a vector of the controls, indexed by the states; $U(x) = a$ is the control applied in state x. The variable $maxIterations$ is first of all there in case the iteration takes too long to converge, and is a hard cap on the number of times the central loop can run. It also lets us run a single step of policy improvement on an arbitrary policy U, by setting $maxIterations = 1$.

function POLICYIMPROVEMENT(U,$maxIterations$) ## *U is the starting policy.*
  $iterationNumber$=0
  $|\mathcal{X}| = \binom{N+K+1}{K+1}$
  $\boldsymbol{\phi} = \mathbf{0}$
  $B = N \max(\mu_i) + \lambda(N)$
  while $iterationNumber < maxIterations$ **do** ## *Run until we hit the limit.*
    $oldU = U$
    $A = \mathbf{0}\mathbf{0}^T$
    $\mathbf{r} = \mathbf{0}$
    for $x \in \mathcal{X}$ **do** ## *Find ϕ for the current policy U.*
      $a = U[x]$
      $\mathbf{r}[x] = R(x+a)/B$
      $A[x,x] = -\frac{1}{B}\lambda((x+a)[K+1])) - \frac{1}{B}(\sum_{k=1}^{K} \mu_k (x+a)[k])$ ## *Self-transition*
      ## *Repair transition - can only happen if we have an asset to repair.*
      if $(x+a)[K+1] > 0$ **then**
        $p_{xy} = \lambda((x+a)[K+1])/B$
        $y = x + a - e_{K+1} + e_{K+2}$
        $A[x,y] = -p_{xy}$
      end if
      for $k \in 1$ to K *s.t.* $(x+a)[k] > 0$ **do**
        ## *Failure transitions - can only happen if there is an asset to fail.*
        $p_{xy} = (x+a)[k]\mu_k/B$
        $y = x + a - e_k + e_{K+1}$
        $A[x,y] = -p_{xy}$
      end for
      $A[x, |\mathcal{X}|+1] = 1$ ## *Average reward.*
    end for
    $A[|\mathcal{X}|+1, 1] = 1$ ## *Fixing condition: $\boldsymbol{\phi}[1] = 0$.*
    $\mathbf{r}[|\mathcal{X}|+1+1] = 0$
    $\boldsymbol{\phi} = A^{-1}\mathbf{r}$
    for $x \in \mathcal{X}$ *s.t.* $x[K+2] > 0$ **do** ## *Solve for the optimal controls, $\forall x$.*
      $val = \mathbf{0}$, of length $|U(x)|$
      for $a \in \mathcal{U}(x)$ **do** ## *For all possible controls.*
        $val[a] = R(x+a) + \boldsymbol{\phi}(x)(B - \lambda((x+a)[K+1]) - \sum_{k=1}^{K} \mu_k (x+a)[k])$
        if $(x+a)[K+1] > 0$ **then**
          $q_{xy} = \lambda((x+a)[K+1])/B$
          $y = x + a - e_{K+1} + e_{K+2}$
          $val[a] = val[a] + q_{xy}\boldsymbol{\phi}[y]$
        end if
        for $k \in 1$ to K *s.t.* $(x+a)[k] > 0$ **do**
          $q_{xy} = (x+a)[k]\mu_k$
          $y = x + a - e_k + e_{K+1}$
          $val[a] = val[a] + q_{xy}\boldsymbol{\phi}[y]$
        end for
      end for
      $U(x) = \arg\max val$
    end for
    if $oldU == U$ **then** ## *Check the termination condition.*
      Return U.
    end if
    $iterationNumber = iterationNumber + 1$
  end while
  Return U.
end function

B.2 Policy Evaluation

We alluded to the approach here in Section 1.2.1, but now we clarify the detail. The two key details are: near-instantanenous transitions under controls, and the use of eigenvalues. it is also worth noting that we do not uniformise the chain, as there is no need.

---

function PolicyEvaluation(U)## *U is the policy to evaluate.*
  $r = \mathbf{0}$ ## *Length is* $|\mathcal{X}|$.
  for $x \in \mathcal{X}$ **do** $r[x] = R(x)$
  end for
  $|\mathcal{X}| = \binom{N+K+1}{K+1}$
  $Q = \mathbf{0}\mathbf{0}^T$ ## *I.e. a matrix of all zeros.*
  $L = 100(N \max(\mu_i) + \lambda(N))$ ## *Near-impulsive controls.*
  for $x \in \mathcal{X}$ **do**
    $a = U[x]$
    ## *We do this differently depending on whether or not*
    ## *the control is non-zero.*
    if $a[k+2] \neq 0$ **then**
      $Q[x, x] = -L$
      $Q[x, x + a] = L$
    else
      Fill in row x of Q as in PolicyImprovement
      $Q[x, x] = -\sum_{y \neq x} Q[x, y]$
    end if
  end for
  Let $\lambda_1, \ldots, \lambda_{|\mathcal{X}|}$ be the eigenvalues of Q
  Find $\mathbf{v}$, the eigenvector corresponding to $\arg\min_{i=1,\ldots,|\mathcal{X}|} |\lambda_i|$
  Normalise $\boldsymbol{\pi} = \mathbf{v}/\|\mathbf{v}\|$
  Return $\boldsymbol{\pi} \cdot \mathbf{r}$.
end function

B.3 Parameter Space Mapping

Suppose we have L different parameters $a^1, a^2, \ldots, a^L$. Each of them takes values in its own (finite) set $A^1, A^2, \ldots A^L$. We want to enumerate all possible combinations, and map them to the integers $1, 2, \ldots$. The essential idea is to use the same integer i modulo the sizes of the various sets to pick which member of the sets we use. Here is the procedure:

---

function PARAMETERSPACE($A^1, A^2, \ldots A^L$; L)
  Find the size $S = \prod_{i=1}^{L} |A^i|$
  Create a matrix R with L columns, and S rows, to hold the combinations
  for i in 1 to S **do**
    $v = i - 1$
    for j in 1 to L **do**
      $w = v \ mod |A^j|$
      $R[i, j] = A^j[w + 1]$
      $v = \lfloor v/|A^j| \rfloor$
    end for
  end for
  Return R.
end function

B.4 State-Space Mapping

As mentioned in Section 4.2 and Section 1.2.1, mapping the state-space $\mathcal{X}$ to the integers $1, \ldots, |\mathcal{X}|$ is nontrivial for many problems, ours included. This is particularly hard because the mapping must be bijective, as if we leave any gaps, when we come to perform any operations on the matrix Q, it will be multiply degenerate.

The solution involves three things, because we want the mapping to be quick each time we use it, but are happy having to spend some time to set up the mapping in the first place. We have a function BackwardsSpaceMap or BSM : $\mathbb{N} \to \mathcal{X}$, a function ForwardsSpaceMap or FSM : $\mathcal{X} \to \mathbb{N}$, which both rely on a matrix *globalLookup* created by a function CreateLookup. The functions BSM and FSM are simple; the function CreateLookup is not and deserves a full explanation.

This is what is invoked in the policy improvement and evaluation algorithms above. When we write **for** $x \in \mathcal{X}$, what that actually means is:

1. **for** i in 1 to $|\mathcal{X}|$
2. $x = \text{BSM}(i)$

In the other direction, when we write $Q[x, y]$, we mean $Q[\text{FSM}(x), \text{FSM}(y)]$. We have left this out of the pseudocode as it is dense enough already.

The key detail for CreateLookup is: we want to generate all vectors x of length $K + 2$ where the components are non-negative integers, and $\sum_{k=1}^{K+2} x_k = N$. We do

this through a form of unrolled recursion, working our way up on K. Additionally, we generate all vectors of length $K+1$ with $\sum_{k=1}^{K+1} x_k \leq N$, and then set $x_{K+2} = N - \sum_{k=1}^{K+1} x_k$ so that the sum is N.

With this complicated setup, we make the resultant matrix *globalLookup* available globally (hence the name), and the resultant maps are simple and simpleish:

```
function BSM(i)
    Return row i of globalLookup.
end function


function FSM(x)
    for i in 1 to |𝒳| do
        Read row i of globalLookup
        if Row i and x are the same then
            Return i.
        end if
    end for
end function
```

---

function CreateLookup(N, K)
  Create a matrix M of with $N+1$ rows and 1 column
  Set $M[i,1] = i-1$ for all possible i
  ## *We have now generated all possible states for 1 task and* $\leq N$ *assets.*
  for j in 2 to $K+1$ **do**
    Create $L = M$, a new matrix
    Define *smallSize* equal to the number of rows in L
    Set $bigSize = 0$
    ## *We now go through each state and see how many*
    ## *new states it will generate when we increase* K *by 1.*
    for i in 1 to *smallSize* **do**
      Set x equal to row i of L
      ## *This is the clever bit. If the vector* x *has* $\sum_k x_k$ *assets,*
      ## *we can tack on any integer* $0, 1, \ldots, N - \sum_k x_k$ *and*
      ## *the new* x *will not have more than* x *assets.*
      Increase *bigSize* by $N + 1 - \sum_k x_k$
    end for
    Recreate M as a matrix with *bigSize* rows and l columns
    Define *place* $= 1$ ## *To keep our place*
    for i in 1 to *smallSize* **do**
      Set x equal to row i of L
      for d in 0 to $N - \sum_k x_k$ **do**
        Set row *place* of H to the vector $(x_1, \ldots, x_l, d)$
        Increment *place* by 1
      end for
    end for
  end for
  ## *We have to deal with the last task seperately, as it needs to*
  ## *take all remaining assets.*
  for i in 1 to $|\mathcal{X}|$ **do**
    Take row i of M
    Append to it $N - \sum_{k=1}^{K+1} M[i,k]$
  end for
  Return M, named *globalLookup*.
end function

B.5 Whittle Indices

In Section 2.4.1 we give a formula for the Whittle indices for the restless bandit approximation of the independent case.:

$$w_k(x_k) = \frac{\sum_{x=0}^{x_k+1} \pi_{x_k+1}(x) g_k(x) - \sum_{x=0}^{x_k} \pi_{x_k}(x) g(x)}{\pi_{x_k}(x_k) - \pi_{x_k+1}(x_k+1)} \tag{B.5.1}$$

Here $\rho_k = \frac{\lambda(x_{K+1}+1)}{\lambda(x_{K+1}+1)+\mu_k}$ and:

$$\pi_{x_k}(x) = \frac{\rho_k^x / x!}{\sum_{y=0}^{x_k} \rho_k^y / y!} \tag{B.5.2}$$

This is an analytic expression, but how do we calculate it efficiently? We can avoid repeated calculation by calculating the probability distributions iteratively, and only normalising at the end. This works as follows:

function WHITTLEINDEX($\mathbf{x}, k, \mu_k$)
$\rho_k = \frac{\lambda(x_{K+1}+1)}{\lambda(x_{K+1}+1)+\mu_k}$
Create vector $\boldsymbol{\pi}^1$ of length $x[k]+2$
$\boldsymbol{\pi}^1[1] = 1$
for i in 1 to $x[k]+1$ **do**
Set $\boldsymbol{\pi}^1[i+1] = \boldsymbol{\pi}^1[i]\frac{\rho_k}{i}$
end for
Create a vector $\boldsymbol{\pi}^0$ of the first $x[k]+1$ elements of $\boldsymbol{\pi}^1$.
$\boldsymbol{\pi}^1 = \frac{\boldsymbol{\pi}^1}{\|\boldsymbol{\pi}^1\|}$
$\boldsymbol{\pi}^0 = \frac{\boldsymbol{\pi}^0}{\|\boldsymbol{\pi}^0\|}$
Create a vector $\mathbf{r}^1$ of length $x[k]+2$
Fill in $\mathbf{r}^1$ as $\mathbf{r}^1[i] = g_k(i)$
Create $\mathbf{r}^0$ out of the first $x[k]+1$ elements of $\mathbf{r}^0$
$top = \boldsymbol{\pi}^1 \cdot \mathbf{r}^1 - \boldsymbol{\pi}^0 \cdot \mathbf{r}^0$
$bottom = \boldsymbol{\pi}^0[x[k]+1] - \boldsymbol{\pi}^1[x[k]+2]$
Return $w = top/bottom$.
end function

Appendix C: Use of Code

The above appendix explains how the algorithms work. But how does the code actually work?

There are two folders "Markovian" and "General", both of which have two folders "Functions" and "Scripts". The folder "General" corresponds to the results Section 3.8, while "Markovian" is responsible for everything else. The folders "Scripts" also contains all of the diagrams and data outputted by the scripts. The contents of "Markovian" and "General" are analogous, so the rest of this should be taken as describing both; we will mention divergences when they arise.

The contents of "Functions" are only ever called, and therefore need only concern you if you are interested in the exact implementation details of the algorithms. In "Markovian/Functions", anything starting "TS_" refers to the two-phase case of Section 3.7.

The folders "Scripts" have the key files in two sets: the scenario-running scripts and the data-extraction scripts. There are also all of the outputs from the scripts, which

have three categories:

1. .txt files, containing LaTeX formatted tables, labelled according to scenario and datapoint. Generated by some of the data-extraction scripts.

2. .pdf files, containing the graphs generated by the data-extraction scripts.

3. .csv files, containing the data from the numerical experiments, generated by the scenario-running scripts.

The data-extraction scripts are largely boring, doing what they describe in their filenames and the comment at the top of each file. The scenario-running scripts are more complex, and we shall explain them now. The pattern is the same for all of them, and is as follows:

1. We load the files from "Functions", and the relevant nonstandard libraries.

2. We set up the sets for the parameter values, as vectors, for example as $N_{vec} = (3, 4, 5, 6, 7, 8, 9, 10)$.

3. We set up dummy values for variables which need to be available globally, not just in specific functions. This includes N, K and μ_k.

4. We set a counter for how many runs we have done already, to allow us to pause and restart computation without losing data.

5. We set up a matrix *experimentData* to hold the data we are about to collect.

6. Now for the big loop. For each scenario, we:

   (a) Decide on the parameters using Appendix B.3.

   (b) For each policy, form it for the current parameters.

   (c) Evaluate each policy with Appendix B.2.

   (d) Write the results and parameters to $experimentData$.

7. Write the matrix $experimentData$ to a suitably named .csv file.

Hopefully this is sufficient explanation. One last thing: all the filepaths are hardcoded, so if running them on a different machine you will have to change them all by hand.

Appendix D: Derivation

This corresponds to the discussion of randomly arriving rewards in 4.5.

Suppose we have a Poisson process A, of rate μ and an exponentially distributed random variable B, of parameter λ. Let N be the number of arrivals to A that happen at a time $t < B$. What is the distribution of N?

Well, the relevant distributions are:

$$Poisson(\mu) \sim e^{-\mu}\frac{\mu^n}{n!} \tag{D.0.1}$$

and

$$Exp(\lambda) \sim \lambda e^{-\lambda t} \tag{D.0.2}$$

So, condition on the event $B = t > 0$. Then we have that

$$p(A = n|B = t) = e^{-\mu t}\frac{\mu^n t^n}{n!} \tag{D.0.3}$$

Now integrate over all possible t:

$$p(N = n) = \int_0^\infty p(A = n|B = t)p(B = t)dt \tag{D.0.4}$$

Plug in the expressions:

$$p(N = n) = \int_0^\infty e^{-\mu t}\frac{\mu^n t^n}{n!}\lambda e^{-\lambda t}dt \tag{D.0.5}$$

This looks like a lot, but we can take everything not involving t safely outside of the integral:

$$p(N = n) = \frac{\mu^n \lambda}{n!}\int_0^\infty t^n e^{-(\lambda+\mu)t}dt \tag{D.0.6}$$

This integral is that of a gamma distribution. Properly, a gamma distribution with parameters α, β is defined over $(0, \infty)$, with distribution:

$$\frac{\beta^\alpha}{\Gamma(\alpha)}x^{\alpha-1}e^{-\beta x} \tag{D.0.7}$$

Comparing to our integral, we have $\alpha = n + 1$ and $\beta = \lambda + \mu$. The integral in Equation (D.0.6) above is therefore equal to the reciprocal of the normalising constant

of that gamma distribution, so that we can write:

$$p(N = n) = \frac{\mu^n \lambda}{n!} \frac{\Gamma(n+1)}{(\lambda+\mu)^{n+1}} \tag{D.0.8}$$

For natural numbers $\Gamma(n) = (n-1)!$. The factor of $n!$ cancels out so that:

$$p(N = n) = \mu^n \lambda \frac{1}{(\lambda+\mu)^{n+1}} = \frac{\lambda}{\lambda+\mu}(\frac{\mu}{\lambda+\mu})^n \tag{D.0.9}$$

This is a geometric distribution, with probability of success $\frac{\mu}{\lambda+\mu}$ $\square$.

Bibliography

[1] S. Ford, M. P. Atkinson, K. Glazebrook, and P. Jacko. On the dynamic allocation of assets subject to failure. *European Journal of Operational Research*, 284(1):227–239, 2020.

[2] N. Nigam. The multiple Unmanned Air Vehicle persistent surveillance problem: A review. *Machines*, 2(1):13–72, 2014.

[3] T.P. Ehrhard. Air force UAVs: The secret history. Technical report, Mitchell Inst for Airpower Studies Arlington VA, 2010.

[4] E.T. Alotaibi, S.S. Alqefari, and A. Koubaa. LSAR: Multi-UAV collaboration for search and rescue missions. *IEEE Access*, 7:55817–55832, 2019.

[5] M. Silvagni, A. Tonoli, E. Zenerino, and M. Chiaberge. Multipurpose UAV for search and rescue operations in mountain avalanche events. *Geomatics, Natural Hazards and Risk*, 8(1):18–33, 2017.

[6] D. Kingston, S. Rasmussen, and L. Humphrey. Automated UAV tasks for search

and surveillance. In *2016 IEEE Conference on Control Applications (CCA)*, pages 1–8. IEEE, 2016.

[7] P. Li and H. Duan. A potential game approach to multiple UAV cooperative search and surveillance. *Aerospace Science and Technology*, 68:403–415, 2017.

[8] The Pacific Ocean. https://www.nationalgeographic.com/environment/oceans/reference/pacific-ocean/, 2020.

[9] H. Gromoll, A. Puha, and R. Williams. The fluid limit of a heavily loaded processor sharing queue. *The Annals of Applied Probability*, 12(3):797–859, 2002.

[10] W. Whitt. Efficiency-driven heavy-traffic approximations for many-server queues with abandonments. *Management Science*, 50(10):1449–1461, 2004.

[11] T.G. Kurtz. Solutions of ordinary differential equations as limits of pure jump Markov processes. *Journal of Applied Probability*, 7(1):49–58, 1970.

[12] S. Davies. UAVs in the firing line. *Engineering & Technology*, 6(8):34–36, 2011.

[13] New Robot. https://xkcd.com/2128/, 2019.

[14] K.E. Jones, K. Dashfield, A.B. Downend, and C.M. Otto. Search-and-rescue dogs: An overview for veterinarians. *Journal of the American Veterinary Medical Association*, 225(6):854–860, 2004.

[15] R. Hassin and M. Haviv. Equilibrium strategies for queues with impatient customers. *Operations Research Letters*, 17(1):41–45, 1995.

[16] L. Merino, F. Caballero, J.R Martínez de Dios, J. Ferruz, and A. Ollero. A cooperative perception system for multiple UAVs: Application to automatic detection of forest fires. *Journal of Field Robotics*, 23(3-4):165–184, 2006.

[17] T. Arnold, M. De Biasio, A. Fritz, and R. Leitner. UAV-based measurement of vegetation indices for environmental monitoring. In *2013 Seventh International Conference on Sensing Technology (ICST)*, pages 704–707. IEEE, 2013.

[18] A. Tripolitsiotis, N. Prokas, S. Kyritsis, A. Dollas, I. Papaefstathiou, and P. Partsinevelos. Dronesourcing: A modular, expandable multi-sensor UAV platform for combined, real-time environmental monitoring. *International journal of Remote Sensing*, 38(8-10):2757–2770, 2017.

[19] M. De Biasio, T. Arnold, and R. Leitner. UAV based multi-spectral imaging system for environmental monitoring. *Technisches Messen*, 78(11):503–507, 2011.

[20] L. F. Gonzalez, G. A. Montes, E. Puig, S. Johnson, K. Mengersen, and K. J. Gaston. Unmanned aerial vehicles (UAVs) and artificial intelligence revolutionizing wildlife monitoring and conservation. *Sensors*, 16(1):97, 2016.

[21] L. Merino, F. Caballero, J.R. Martínez-De-Dios, I. Maza, and A. Ollero. An unmanned aircraft system for automatic forest fire monitoring and measurement. *Journal of Intelligent & Robotic Systems*, 65(1-4):533–548, 2012.

[22] H. Kerzner. *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*. John Wiley & Sons, 10th edition, 2017.

[23] J.R. Meredith, S.M. Shafer, and S.J. Mantel Jr. *Project Management: A Strategic Managerial Approach*. John Wiley & Sons, 2017.

[24] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[25] Y. Qiu and J. Mei. *RSpectra: Solvers for Large-Scale Eigenvalue and SVD Problems*, 2019. R package version 0.16-0.

[26] A. Condon and M. Melekopoglou. On the complexity of the policy improvement algorithm for Markov decision processes. *ORSA Journal on Computing*, 6(2):188–192, 1994.

[27] F. Dufour and A. B. Piunovskiy. Impulsive control for continuous-time Markov decision processes. *Advances in Applied Probability*, 47(1):106–127, 2015.

[28] J. C. Gittins, K. Glazebrook, and R. Weber. *Multi-armed Bandit Allocation Indices*. John Wiley & Sons, 2011.

[29] J. C. Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society. Series B (Methodological)*, 41(2):148–164, 1979.

[30] D.A. Berry and B. Fristedt. Bandit problems: Sequential allocation of experiments. *London: Chapman and Hall*, 5(71-87):7–7, 1985.

[31] P. Whittle. Restless bandits: Activity allocation in a changing world. *Journal of Applied Probability*, 25(A):287–298, 1988.

[32] K. Glazebrook. D. Ruiz-Hernandez and C. Kirkbride. Some indexable families of restless bandit problems. *Advances in Applied Probability*, 38(3):643–672, 2006.

[33] R. Weber and G. Weiss. On an index policy for restless bandits. *Journal of Applied Probability*, 27(3):637–648, 1990.

[34] Resource Allocation. https://www.britannica.com/topic/operations-research/Resource-allocation, 2020.

[35] R.Z. Farahani, M. Steadie-Seifi, and N. Asgari. Multiple criteria facility location problems: A survey. *Applied Mathematical Modelling*, 34(7):1689–1709, 2010.

[36] D.B. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 265–274, 1997.

[37] D.J. Hoitomt, P.B. Luh, and K.R. Pattipati. A practical approach to job-shop scheduling problems. *IEEE Transactions on Robotics and Automation*, 9(1):1–13, 1993.

[38] Eric E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[39] T. Yamada and R. Nakano. Genetic algorithms for job-shop scheduling problems. In *Proceedings of Modern Heuristics for Decision Support*, pages 67–81. UNICOM Seminar London, 1997.

[40] D. Pisinger. An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114(3):528–541, 1999.

[41] C.E. Ferreira, A. Martin, and R. Weismantel. Solving multiple knapsack problems by cutting planes. *SIAM Journal on Optimization*, 6(3):858–877, 1996.

[42] M.S. Hung and J.C. Fisk. An algorithm for 0-1 multiple-knapsack problems. *Naval Research Logistics Quarterly*, 25(3):571–579, 1978.

[43] N. Sundarapandian. *Probability, Statistics and Queuing Theory*. PHI Learning Pvt. Ltd., 2009.

[44] K. Avrachenkov, U. Ayesta, J. Doncel, and P. Jacko. Congestion control of TCP flows in internet routers by means of index policy. *Computer Networks*, 57(17):3463–3478, 2013.

[45] A. Asadi, P. Jacko, and V. Mancuso. Modeling D2D communications with LTE and WiFi. *ACM SIGMETRICS Performance Evaluation Review*, 42(2):55–57, 2014.

[46] I. Taboada, P. Jacko, U. Ayesta, and F. Liberal. Opportunistic scheduling of flows with general size distribution in wireless time-varying channels. In *2014 26th International Teletraffic Congress (ITC)*, pages 1–9. IEEE, 2014.

[47] U. Ayesta, M. Erausquin, and P. Jacko. A modeling framework for optimizing the flow-level scheduling with time-varying channels. *Performance Evaluation*, 67(11):1014–1029, 2010.

[48] J.F.C. Kingman. The first Erlang century—and the next. *Queueing Systems*, 63(1-4):3, 2009.

[49] S.K. Bose. *An introduction to queueing systems*. Springer Science & Business Media, 2013.

[50] L. Lakatos, L. Szeidl, and M. Telek. *Introduction to Queueing Systems with Telecommunication Applications*. Springer, 2019.

[51] R. Shone, V.A. Knight, and P.R. Harper. A conservative index heuristic for routing problems with multiple heterogeneous service facilities. *Mathematical Methods of Operations Research*, 92(3):511–543, 2020.

[52] R. Shone, K. Glazebrook, and K.G. Zografos. Resource allocation in congested queueing systems with time-varying demand: An application to airport operations. *European Journal of Operational Research*, 276(2):566–581, 2019.

[53] J-C. Ke. The optimal control of an M/G/1 queueing system with server vacations, startup and breakdowns. *Computers & Industrial Engineering*, 44(4):567–579, 2003.

[54] O. Kella. The threshold policy in the M/G/1 queue with server vacations. *Naval Research Logistics (NRL)*, 36(1):111–123, 1989.

[55] A. Federgruen and K. C. So. Optimality of threshold policies in single-server queueing systems with server vacations. *Advances in Applied Probability*, 23(2):388–405, 1991.

[56] R. Shone, K. Glazebrook, and K.G. Zografos. Stochastic modelling of aircraft queues: A review. In *60th Annual Conference of the Operational Research Society*, pages 61–83, 2018.

[57] W. Ren and Y. Cao. *Distributed coordination of multi-agent networks: Emergent problems, models, and issues*. Springer Science & Business Media, 2010.

[58] A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyre. Recent advances on multi-agent patrolling. In *Brazilian Symposium on Artificial Intelligence*, pages 474–483. Springer, 2004.

[59] M. Othmani-Guibourg, A. El Fallah-Seghrouchni, J-L. Farges, and M. Potop-Butucaru. Multi-agent patrolling in dynamic environments. In *2017 IEEE international conference on agents (ICA)*, pages 72–77. IEEE, 2017.

[60] P.B. Sujit and D. Ghose. Search using multiple UAVs with flight time constraints. *IEEE Transactions on Aerospace and Electronic Systems*, 40(2):491–509, 2004.

[61] C. Erignac. An exhaustive swarming search strategy based on distributed pheromone maps. In *AIAA Infotech@ Aerospace 2007 Conference and Exhibit*, page 2822, 2007.

[62] J. Redding, T. Toksoz, N.K. Ure, A. Geramifard, J.P. How, M.A. Vavrina, and J. Vian. Distributed multi-agent persistent surveillance and tracking with health management. In *Proceedings of the AIAA Guidance Navigation and Control Conference, Portland, OR, USA*, pages 8–11, 2011.

[63] X. Zheng, S. Jain, S. Koenig, and D. Kempe. Multi-robot forest coverage. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3852–3857. IEEE, 2005.

[64] G. Max. Modelling computer networks. In *Journal of Physics: Conference Series*, volume 268, pages 12–18. IOP Publishing, 2011.

[65] T.M. Chen. Network traffic modeling. In *The Handbook of Computer Networks*, volume 3, page 156. Wiley Hoboken, NJ, 2007.

[66] U. Akpan, C. Kalu, S. Ozuomba, and A. Obot. Development of improved scheme for minimising handoff failure due to poor signal quality. *International Journal of Engineering Research and Technology*, 2(10):2764–2771, 2013.

[67] N. Zong and M. Dhanasekar. Sleeper embedded insulated rail joints for minimising the number of modes of failure. *Engineering Failure Analysis*, 76:27–43, 2017.

[68] S.-H. Ding and S. Kamaruddin. Maintenance policy optimization—literature review and directions. *The International Journal of Advanced Manufacturing Technology*, 76(5-8):1263–1283, 2015.

[69] L. Rhodes-Leader, B.S.S. Onggo, D.J. Worthington, and B.L. Nelson. Multi-fidelity simulation optimisation for airline disruption management. In *2018 Winter Simulation Conference (WSC)*, pages 2179–2190. IEEE, 2018.

[70] D. Bergemann and J. Valimaki. Bandit problems. 2006.

[71] A. Mahajan and D. Teneketzis. Multi-armed bandit problems. In *Foundations and Applications of Sensor Management*, pages 121–151. Springer, 2008.

[72] T. Lattimore and C. Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

[73] K. Glazebrook, C. Kirkbride, and J. Ouenniche. Index policies for the admission control and routing of impatient customers to heterogeneous service stations. *Operations Research*, 57(4):975–989, 2009.

[74] J. Niño-Mora. Dynamic priority allocation via restless bandit marginal productivity indices. *TOP*, 15(2):161–198, 2007.

[75] P. Nesbitt, Q. Kennedy, J. K. Alt, and R. Fricker. Iowa gambling task modified for military domain. *Military Psychology*, 27(4):252–260, 2015.

[76] J. Marshall. *Models of intelligence operations*. PhD thesis, Lancaster University, 2016.

[77] K. Glazebrook, C. Kirkbride, H.M. Mitchell, D.P. Gaver, and P.A. Jacobs. Index policies for shooting problems. *Operations Research*, 55(4):769–781, 2007.

[78] A.O. Hall. Military operations research: A career's worth of opportunities in the US army. In *Pushing the Boundaries: Frontiers in Impactful OR/OM Research*, pages 106–127. INFORMS, 2020.

[79] K. Glazebrook, C. Kirkbride, and D. Ruiz-Hernandez. Spinning plates and

squad systems: Policies for bi-directional restless bandits. *Advances in Applied Probability*, 38(1):95–115, 2006.

[80] I.M. Verloop et al. Asymptotically optimal priority policies for indexable and nonindexable restless bandits. *The Annals of Applied Probability*, 26(4):1947–1995, 2016.

[81] U. Ayesta, M.K. Gupta, and I.M. Verloop. On the computation of Whittle's index for Markovian restless bandits. *Mathematical Methods of Operations Research*, pages 1–30, 2020.

[82] Z. Yu, Y. Xu, and L. Tong. Deadline scheduling as restless bandits. *IEEE Transactions on Automatic Control*, 63(8):2343–2358, 2018.

[83] J. Nino-Mora. Computing a classic index for finite-horizon bandits. *INFORMS Journal on Computing*, 23(2):254–267, 2011.

[84] T. James. *Control of multiclass queueing systems with abandonments and adversarial customers*. PhD thesis, Lancaster University, 2016.

[85] U. Ayesta, P. Jacko, and V. Novak. Scheduling of multi-class multi-server queueing systems with abandonments. *Journal of Scheduling*, 20(2):129–145, 2017.

[86] A. Abbou and V. Makis. Group maintenance: A restless bandits approach. *INFORMS Journal on Computing*, 31(4):719–731, 2019.

[87] M. Larrañaga. *Dynamic control of stochastic and fluid resource-sharing systems*. PhD thesis, INP Toulouse, 2015.

[88] M. Harchol-Balter. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013.

[89] S. Stidham and R. Weber. A survey of Markov decision models for control of networks of queues. *Queueing Systems*, 13(1-3):291–314, 1993.

[90] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. John Wiley & Sons, 2007.

[91] K. Lin, M. Atkinson, and K. Glazebrook. Optimal patrol to uncover threats in time when detection is imperfect. *Naval Research Logistics*, 61(8):557–576, 2014.

[92] K. L. B. Cook. The silent force multiplier: The history and role of UAVs in warfare. In *2007 IEEE Aerospace Conference*, pages 1–7, 2007.

[93] S. Waharte and N. Trigoni. Supporting search and rescue operations with UAVs. In *International Conference on Emerging Security Technologies (EST)*, pages 142–147, 2010.

[94] D. W. Casbeer, R. W. Beard, T. W. McLain, S-M. Li, and R. K. Mehra. Forest fire monitoring with multiple small UAVs. In *Proceedings of the 2005 American Control Conference*, pages 3530–3535, 2005.

[95] B. Boyacı, K. G. Zografos, and N. Geroliminis. An optimization framework for the development of efficient one-way car-sharing systems. *European Journal of Operational Research*, 240(3):718–733, 2015.

[96] J. Schuijbroek, R.C. Hampshire, and W.-J. Van Hoeve. Inventory rebalancing and vehicle routing in bike sharing systems. *European Journal of Operational Research*, 257(3):992–1004, 2017.

[97] B. T. Doshi. Queueing systems with vacations: A survey. *Queueing Systems*, 1:29–66, 1986.

[98] E. Hyytiä. Optimal routing of fixed size jobs to two parallel servers. *INFOR: Information Systems and Operational Research*, 51(4):215–224, 2013.

[99] V. S. Borkar and S. Pattathil. Whittle indexability in egalitarian processor sharing systems. *Annals of Operations Research*, Online First, 2017.

[100] J. Nino-Mora. Restless bandits, partial conservation laws and indexability. *Advances in Applied Probability*, pages 76–98, 2001.

[101] M. Flint, Emmanuel E. Fernandez, and M. Polycarpou. Stochastic models of a cooperative autonomous UAV search problem. *Military Operations Research*, pages 13–32, 2003.

[102] L. Bertuccelli, W. Beckers, and M. Cummings. Developing operator models for UAV search scheduling. In *AIAA Guidance, Navigation, and Control Conference*, page 7863, 2010.

[103] P. Doherty and P. Rudol. A UAV search and rescue scenario with human body detection and geolocalization. In *Australasian Joint Conference on Artificial Intelligence*, pages 1–13, 2007.

[104] S.K. Dhall and C.L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, 1978.

[105] L. Schmickler. MEDA: mixed Erlang distributions as phase-type representations of empirical distribution functions. *Communications in Statistics: Stochastic Models*, 8(1):131–156, 1992.

[106] V. Krishnamurthy. *Partially Observed Markov Decision Processes*. Cambridge University Press, 2016.

[107] R.S. Sutton and A.G. Barto. *Reinforcement learning : an introduction*. MIT Press, 1998.

[108] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 2017.

[109] M. Koes, I. Nourbakhsh, K. Sycara, and S.D. Ramchurn et. al. Heterogeneous multirobot coordination with spatial and temporal constraints. In *AAAI*, volume 5, pages 1292–1297, 2005.

[110] K. Nakade and R. Nishiwaki. Optimal allocation of heterogeneous workers in a U-shaped production line. *Computers & Industrial Engineering*, 54(3):432–440, 2008.

[111] W.B. Powell, J.A. Shapiro, and H.P. Simão. An adaptive dynamic programming algorithm for the heterogeneous resource allocation problem. *Transportation Science*, 36(2):231–249, 2002.

[112] S. Vakili, K. Liu, and Q. Zhao. Deterministic sequencing of exploration and exploitation for multi-armed bandit problems. *IEEE Journal of Selected Topics in Signal Processing*, 7(5):759–767, 2013.

[113] J.-Y. Audibert, R. Munos, and C. Szepesvári. Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19):1876–1902, 2009.

[114] N. Galichet, M. Sebag, and O. Teytaud. Exploration vs. exploitation vs. safety: Risk-aware multi-armed bandits. In *Asian Conference on Machine Learning*, pages 245–260, 2013.

[115] K. Glazebrook. On a class of non-Markov decision processes. *Journal of Applied Probability*, 15(4):689–698, 1978.

[116] S.D. Whitehead and L.-J. Lin. Reinforcement learning of non-Markov decision processes. *Artificial Intelligence*, 73(1-2):271–306, 1995.

[117] D. Bertsekas. *Dynamic programming and optimal control*. Athena Scientific Belmont and MA, 1995.

[118] W.B. Powell. What you should know about approximate dynamic programming. *Naval Research Logistics*, 56(3):239–249, 2009.

[119] C. Pike-Burke, S. Agrawal, C. Szepesvari, and S. Grunewalder. Bandits with delayed, aggregated anonymous feedback. In *International Conference on Machine Learning*, pages 4105–4113, 2018.

[120] Z. Zhou, R. Xu, and J. Blanchet. Learning in generalized linear contextual bandits with stochastic delays. In *Advances in Neural Information Processing Systems*, pages 5197–5208, 2019.

[121] J. Tyo, O. Neopane, J. Byrd, C. Gupta, and C. Igoe. Multi-armed bandits with delayed and aggregated rewards. Technical report, CCDC ARL Adelphi United States, 2019.

[122] T.L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.