Implementation and Analysis of Approximation Algorithms For One-Counter Markov Decision Processes

Shreyas Saxena

4th Year Project Report
Computer Science and Mathematics
School of Informatics
University of Edinburgh

2021

Abstract

We implement and analyze several algorithms of some key analysis problems for One-Counter Markov Decision Processes (OC-MDPs), a type of finitely-presented infinite state MDPs.

One-Counter Markov Decision Processes are a mathematical model inspired from Quasi Birth Death processes, a randomized model studied in stochastic modelling theory and applied probability. OC-MDPs have probabilistic and control states, where every probabilistic state defines a probability distribution over the set of its outgoing transition, while in control states, the agent is required to choose a particular action which dictates which probabilistic states he/she will visit. OC-MDPs also have an unbounded counter which can be interpreted at the total reward the player has achieved at a given time step. At every control state, an agent must pick between its allowed number of actions in order to achieve a certain objective. Examples of such objectives include the long-run average reward obtained per transition, and covering very low total reward values in the long-run.

In this study we implement algorithms in Python to approximate the optimal probability of the agent "terminating" given that (s)he starts above a counter value N. That is, if the agent starts with a given counter value $j > N$, our algorithms are able to approximate the probability that the agent ensures that the counter value (the total reward) will hit 0 upto a certain error ε. We also implement algorithms to compute an upper bound on the counter value N and illustrate the challenges behind implementing these algorithms.

We also empirically analyze the performance of these algorithms by experimenting them on a well known subclass of OC-MDPs known as solvency games. Solvency games are extremely relevant to mathematical finance as they can model the behaviour of risk-averse investors. Our results show that the upper bound on the counter value N is not tight enough and we provide a heuristic that allows one to calculate a tighter bound on the counter value than currently implemented. We also show that some of the subproblems used to approximate the optimal termination probabilities can be greatly simplified for an average OC-MDP.

Our algorithms combine theory from linear programming, stochastic modelling, graph theory, and MDP reward models.

Acknowledgements

I would like to thank my supervisor Professor Kousha Etessami for tremendous help in this project and guidance in talking to the right people and reading the right papers. I would also like to thank my friends and family for their continued support in writing this dissertation and providing valuable emotional advice during these tough and uncertain times.

Table of Contents

Chapter 1

Introduction

1.1 One-Counter Markov Decision Processes (OC-MDPs)

Markov Decision Processes (MDPs) are discrete-time stochastic control processes, where the outcome can either be random or in the control of a decision maker, or *agent*. MDPs tend to begin in some state and then make a sequence of transitions. These transitions can either be made by the agent, or they can be purely random based on a predefined probability distribution over possible transitions. The transition graphs of such stochastic models arise from more familiar models such as context-free, counter, and pushdown automaton (see [5]). Our concern with MDPs lies specifically with respect to the agent's *objective*, or goal. The agent's goal is usually to optimize the expected value of an objective function, which is usually the function of a trajectory. In order to achieve this goal, one can fix a strategy for the controller which defines a Markov chain defined by which actions to take at each control state, defining a probability space of possible runs, or sequences of states visited. From a computational standpoint, the biggest questions that arise are the computation complexities of problems concerning the optimal value a controller can achieve for a fixed objective and the strategies that can achieve this value. For finite-state MDPs, these problems have been extensively studied for many objectives, and there are many papers that suggest methods that work well.

In this study we focus on infinite state stochastic processes which are presented finitely through the use of an unbounded counter. Examples include branching and birth-death processes. There are several well-studied models that exhibit similar behaviour to these types of processes. One most relevant to our study is Quasi-Birth-Death processes (QBDs), which can be viewed as an unbounded queue that has a counter tracking the number of "jobs" and can belong to an element in a bounded set of "states". A stochastic transition tends to involve adding or removing jobs from this queue or transitioning the queue from one state to another.
A flavor of QBDs known as embedded discrete-time QBDs are more relevant to our task. These discrete-time QBDs can be equivalently viewed as a probabilistic extension of counter automaton which are ex-

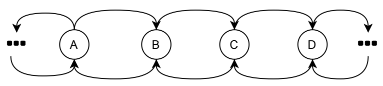

Figure 1.1: Example of a Quasi Birth Death Process

tensions of classic finite-state machines with an unbounded counter, which can be incremented, decremented, or stay the same with each transition. Furthermore, transitions themselves can also be dependent upon a combination of the current state and the counter value. Thus, combining QBDs with a counter with a controller yields us to the main model of interest: **One-Counter Markov Decision Processes (OC-MDPs)**. OC-MDPs have a finite set of random states, where the next transition is chosen subject to a defined probability distribution, and a finite set of controlled states, where the next transition is chosen by the agent. OC-MDPs have "configurations" which consists of a pair (s, j) where s is a state of the OC-MDP and j is the current counter value. As with MDPs, OC-MDPs too give rise to different types of computational problems and objectives, covered in Section 1.2.

To explain some of the applications of this project, we turn to an example of a subset of OC-MDPs whose computational objectives have been studied called solvency games, defined in [2] as a mathematical model of risk-averse investors. Solvency games are a subclass of OC-MDPs with only one control state and a more complex transition function. At every time step, there are multiple actions that can change the counter value (or "wealth"), by at most 1 per transition. The goal in this game is to minimize the probability of going bankrupt, starting with a given positive wealth. [2] shows that if the solvency games meet certain assumptions then the best choice of an agent once his/her wealth is above a certain threshold can be computed in exponential time through linear programming. This strategy, however, is not always optimal, and [2] concedes that the "results are at best a sketch of some elements of a larger theory", which is further discussed in [6] and Section 1.2. In Chapter 5, we experiment with a realistic example of modified version of a solvency game to try to compute the investor's (or its adversary's) best choice to ensure bankruptcy by rephrasing the problem as one of termination starting from a given wealth.

1.2 Termination Probabilities

The computation problem and objective that we focus on is known as the *termination* objective and its associated "termination probabilities". In OC-MDPs, the player aims to maximize (minimize) the probability of eventually hitting counter value 0 (in any control state), given that they start at a control state with a given counter value $j > 0$. Current research is mostly concerned with trying to maximize the termination probability for a "boundaryless" OC-MDP, where the counter value can take on both positive and negative values. These kinds of analysis problems lend themselves into two categories: *quantitative* and *qualitative* analysis. Both are concerned with if a given probability p acts as a lower or upper bound for the probability of the objective being achieved. While quantitative analysis considers $p \in [0, 1]$, qualitative analysis is strictly concerned with $p \in \{0, 1\}$. Research has also been made into the type of strategies that achieve these outcomes for OC-MDPs.

For OC-MDPs, [6] first devises algorithms for maximizing the termination probability in a boundaryless OC-MDP, with the objective of optimizing the probability that the lim inf counter value is $-\infty$ (the paper describes this objective as "Cover Negative"

(CN)). Notes that this model is related to that of finite-state MDPs with objectives concerning the average reward, and [6] utilizes this connection to compute the optimal termination probabilities for the Cover Negative objective. More specifically, they show that the probability that the lim inf of the counter value is $-\infty$ for a boundaryless OC-MDP is a rational value that can be computed in **P**-time, and that the optimal deterministic strategy is *"counter oblivious"* (same for every counter value) and *"memoryless"* (doesn't depend on the previous moves taken by the agent) and can be computed in **P**-time. [6] shows this by first showing that the Cover Negative problem for MDPs can be reduced to its qualitative version, and then showing that this qualitative version can be further reduced to a qualitative average reward problem (this problem is formally defined in Chapter 2 as a Mean Payoff problem) which can be solved in polynomial time through an algorithm known as `Qual-MP`. With `Qual-MP`, [6] develops algorithms to solve the Qualitative and Quantitative Cover Negative problem by implementing this algorithm and other adjustments. Further explanations are detailed in future sections. This paper is tantamount in laying the foundations of algorithmic OC-MDP theory and the algorithms discussed in [6] will be the main focus of the algorithms that we will implement in this project. However, the views discussed in [6] are still elementary at best, and only draw a sketch of a larger, more complex problem.

An important notion for any objective is a game's *value*, which is defined in [6] as the number v such that for every $\varepsilon > 0$, the agent has a strategy that ensures that the objective is satisfied with probability at least $v - \varepsilon$ regardless of the outcomes of the probabilistic states. In this paper we use optimal probability and value interchangeably. The authors of [6] build on the theory of their previous paper in [4] where they state that for both limit objectives, $LimInf(= -\infty) = LimSup(= \infty)$ and $LimInf(= \infty)$, given a finite-state OC-MDP $\mathcal{D}$ with rewards and a probability p such that $p \in \mathbb{Q}$, deciding whether the value of $\mathcal{D}$ with either limit objective is $\leq p$ or $\geq p$ can be solved in **P**-time. The exact proof of these theorems is not relevant to this project. It is also shown that if the value of the termination game is 1 then Max has an optimal memoryless counter-oblivious, and pure strategy to ensure termination with probability converging to 1 [4]. These strategies can be computed in **P**-time.

As shown earlier, [4] dealt with the qualitative termination problem of deciding if the termination value is 1. The quantitative problem for OC-MDPs, however, had been left open for investigation. This was resolved in [5], with a concession that the termination value can only be approximated. This is due to several reasons. For maximizing OC-MDPs, there doesn't have to be any optimal strategy for maximizing the termination probability, only ε-optimal ones. For minimizing OC-MDPs, the strategies do exist but they are very complicated to compute. Thus, approximating the termination value and computing ε-optimal strategies simplifies the problem significantly. The main objective of [5] is to show that there exists an algorithm that takes the following as an input: an OC-MDP, $\mathcal{D}$, an initial control state s, an initial counter value $j > 0$ and a rational threshold ε. Given these inputs the algorithm computes the following:

- a $v' \in \mathbb{Q}$ such that $|v' - v^*| < \varepsilon$, where v^* is the value of the OC-MDP termination game on $\mathcal{D}$ starting from state (s, j)
- ε-optimal strategies for the agent in the OC-MDP.

[5] shows that for OC-MDPs, said algorithm runs in exponential time in the encoding size of the OC-MDP and polynomial time in $\log(\varepsilon^{-}1)$ and $\log(j)$. What is important to note from the derivation of this algorithm is that for *maximizing* OC-MDPs where the player is trying to prevent ending up with a counter value of 0 , it is possible to compute a bound N on the counter value such that for all counter values $n > N$, the optimal termination probability starting at state (q,n) is at most ε away from the optimal probability for the counter to have lim inf value = $-\infty$. This bound allows the problem to be reduced to a reachability problem for a much (exponentially) larger finite-state MDP which can be solved through linear programming in exponential time.

1.3 Aim of Project

So far current literature has extensively discussed the theoretical foundations and outlined the generic instructions for some of these algorithms that compute the optimal strategies, and termination values and probabilities for these objectives and stochastic games. There is, however, no record of these algorithms being implemented and analyzed for their runtime yet. While there are probabilistic model checking tools such as PRISM which may appear to simplify this analysis significantly, a conversation with the creator of this application, Dave Parker, revealed that the software doesn't support analysis of infinite horizon average reward objectives which are central to this paper. Outside of PRISM, there are tools available for the analysis of MDPs for data science applications such as reinforcement learning, but there is not much publicly available that allows for a game theoretic analysis of OC-MDPs. This paper aims to implement some of the algorithms for qualitative analysis and approximation of termination value. The theory used to develop these algorithms has also made several statements that have complicated the problem to take care of pathological edge cases, and thus another aim for this project is to look for potential points of inefficiencies, apply these algorithms to a potential use case of Markov Decision Processes which illustrates the most usual case of MDPs that will be used in the real world, and suggest the necessity to revisit this theory to check for improvements.

Chapter 2

Definitions

This honours project focuses on the study of Markov Decision Processes, which in itself is a rather complicated mathematical field. In this section, we shall formally define their mathematical definitions and related concepts.

Definition 1. *A **Markov Decision Process (MDP) (with rewards)** is a tuple $\mathcal{M} = (S, \Delta, (S_N, S_p), p, r)$, where the following hold true:*

- *S is the* state space*, a finite, countable set of vertices*
- *Δ is the* action space*, where $\Delta \subseteq S \times S$ which defines the transitions from one state in S to another in the same set. Note that $\forall v \in V, \exists u \in V$, where $v \rightarrow u$ (i.e. there are no states with no outgoing transitions).*
- *S_N, S_P is the* partition *of S into control and probabilistic states.*
- *p is a* probability assignment *such that each probabilistic vertex $v \in S_P$ is assigned a rational probability distributions on its set of outgoing transitions.*
- *(only in an MDP with rewards) r is the reward function $r : V \rightarrow \mathbb{R}$.*

As is the case with many stochastic models, we can also establish the following terminology which may prove useful in the future. A *path w* in MDP $\mathcal{M}$ is a finite or infinite sequence of vertices, i.e. $w = w(0)w(1)w(2)...w(n-1)$, such that $w(i-1) \rightarrow w(i)$ for all $1 \leq i \leq len(w)$, where $len(w) \in [0, \infty]$ is the length of the sequence of vertices. A *run* is simply an infinite path. The set of all runs in $\mathcal{M}$ is denoted by $Run_{\mathcal{M}}$, and the set of all runs starting with some finite path w is denoted by $Run_{\mathcal{M}}(w)$, and these sets create the standard Borel Sigma Algebra on $Run_{\mathcal{M}}$ [6].

Having defined MDPs, we now extend the definition to involve an unbounded counter, leading us to the concept of One-Counter Markov Decision Processes.

Definition 2. *A **One-Counter Markov Decision Process (OC-MDP)** is a tuple $\mathcal{A} = (Q, \delta^{=0}, \delta^{>0}, \delta^{=0}, (Q_N, Q_P), P^{>0}, P^{=0})$ where the following hold true*

- *Q is a finite set of states partitioned into non-deterministic, control states Q_N and probabilistic states Q_P*

- $\delta^{=0} \subseteq Q \times \{-1,0,1\} \times Q$ *and* $\delta^{=0} \subseteq Q \times \{0,1\} \times Q$ *are the sets of* positive *and* zero rules *(transitions), respectively*
- $P^{>0}$ *and* $P^{>0}$ *are the probability assignments such that both assign a positive rational probability distribution over outgoing transitions in* $\delta^{>0}$ *and* $\delta^{=0}$ *(respectively) for each probabilistic state p.*

For OC-MDPs, we are interested in computing some of their *strategies*. To define a strategy we first note that a *configuration* is a pair (q,c) of control states q and integer counter value $c \in \mathbb{Z}$. Given this we can now define a strategy for the agent.

Definition 3. *A* policy (or strategy) *for the agent is a function which, to each state, assigns a probability distribution on the different actions available to the state. They act as the optimal probability of the player picking each action at the given state. A* deterministic *strategy is when the probability distribution on each state is a* Dirac delta function *centered at a particular action. They are similar to the notion of* pure *strategies in game theory.*

For finite-state MDPs, we represent policies as tuples. So for example, in a 5-state MDP with at most 3 actions in each state, an example policy for an objective can be: $(0,2,1,1,2)$ (if zero-based indexing is used). This means that at the 0th state, player must take action 0 **with probability 1**, at the 1st state, player must take action 2 **with probability 1**, and so on and so forth.

Finally, in order to devise a policy that can meet these objectives, we consider something known as a *value function*. The notion of such a function is central to our analysis of MDPs as these policies are formed by attempting to optimize these value functions.

Definition 4. *Given a policy* π*, and a state* $s \in V$*, the* value function $V^{\pi}(s)$ *for the state s is defined as the expected return when starting in s and following* π *thereafter. For our analysis of MDPs, we can define the value function formally as:*

$$V^{\pi}(s) = \mathbb{E}_{\pi}\{R_t \mid s_t = s\}$$

where $\mathbb{E}_{\pi}$ *denotes the expected value (or reward) given that the agent follows policy* π *and t is any time step (including* $t = 0$*).*

This definition is very important because it implies that when computational methods such as Linear Programming, Policy Iteration, and Value Iteration are applied to MDPs, the values of the value function we get will detail the expected reward of the policy starting **from each given state**. This is further echoed in [7], which states that the optimal policy produced maximizes the value function **for all states**, i.e. $\forall s \in V, V^{\sigma}(s) \leq V^{\pi}(s)$, where π is the optimal policy and σ is any other arbitrary policy.

Note that these policies are aimed at achieving a certain objective. In this paper, we will implement algorithms aimed at solving several objectives which are defined below.

Definition 5. *Consider an infinite time horizon*[1]*. The* mean payoff objective *is the objective where the agent seeks to minimize/maximize the expected average reward*

[1] we are playing for an infinite duration of time

over this infinite time horizon. This is a way of describing the long-run limiting average reward of a finite state MDP with a given reward function. Formally, we represent this as

$$\lim_{x\to\infty} \mathbb{E}_\pi \left[\frac{1}{n} \sum\nolimits_{t=0}^{n-1} [R_t \mid s_0] \right]$$

In this project we will not implement algorithms to maximize the mean payoff. Thus for the rest of this paper, when we refer to mean payoff from here on we refer to minimizing the expected average reward over the infinite time horizon.

We also define the Cover Negative objective:

Definition 6. *The cover negative (or limit) objective, is defined as the objective where the agent seeks to make sure that the counter value during the run covers arbitrarily low negative numbers in $\mathbb{Z}$. If we define a sequence of vertices as a run, we formally define the limit objective by stating that it is the set of runs where the following property holds true:*

$$\mathit{LimInf}(= -\infty) = \{w \mid w \textit{ is a run in such that } \liminf_{n\to\infty} \sum\nolimits_{i=0}^{n} r(w(i)) = \infty\}$$

[4]

This objective can be redefined for different variations such as $\mathit{LimInf}(=\infty)$, $\mathit{LimSup}(=\infty)$, $\mathit{LimSup}(=-\infty)$. We can say that a given policy meets the defined objective if the policy-induced Markov Chain has some runs, or sequence of states, which belong to this set.

The mean-payoff objective is necessary when trying to compute a strategy that meets the cover negative objective. As we will show while explaining the algorithms implemented in Section 3, we can modify the MDP in a certain way so as to ensure that a strategy that meets the Mean Payoff objective also meets the Cover Negative Objective. Our main objective of interest, however, is the termination objective. By solving the Cover Negative objective, we can develop ε-optimal strategies for this termination objective for all initial counter values above a pre-computed N [5]:.

Definition 7. *Given an initial counter value j, we say that a given a run, or sequence of vertices, satisfies the* termination objective *if the run belongs to the set $Term(j)$, where $Term(j)$ is defined as follows:*

$$\begin{aligned} Term(j) = \{w \mid\ & w \textit{ is a run in the OC-MDP such} \\ & \textit{that there exists a finite } m > 0 \textit{ such that } \sum\nolimits_{i=0}^{n} r(w(i)) = -j\} \end{aligned}$$

Developing subroutines that allow us to compute the optimal strategy for termination will be the focus of our next few sections, after which we will perform several experiments to test its performance in a certain type of MDP.

Chapter 3

Algorithms to Implement

In this section, we explain the theoretical underpinnings of the algorithms which will allow us to approximate the termination value for one-counter Markov Decision Processes. For OC-MDPs, there exists a counter value N such that for all configurations with counter value greater than or equal to N, the ε-optimal termination probabilities are equivalent to those for the cover negative objective [5]. Solving the cover negative objective itself, however, is a highly nontrivial task. We start by explaining how the problem of the cover negative objective can be resolved to several subproblems, and how those individual subproblems are satisfied. Then, we shall analyze how the cover negative problem is used to solve the termination objective for OC-MDPs.

Recall from the previous chapter that we defined the cover negative problem to be the objective where the agent wishes to ensure that the counter value covers arbitrarily low negative numbers in $\mathbb{Z}$. [6] shows that for any OC-MDP, there exists a optimal counterless, memoryless, deterministic policy that is computable in polynomial time. Furthermore, the value of the game with respect to this objective is also computable in polynomial time. To prove the existence of such a policy, they make 4 key Turing Reductions, reducing the problem of cover negative into the mean payoff problem for finite state MDPs with rewards. This is achieved by modifying the OC-MDP. The general idea behind these reductions is that the qualitative cover negative problem for an MDP can be reduced to the qualitative mean payoff problem for a modified MDP. From there, as this chapter will explain, we can make new MDPs off the current one to solve the cover negative problem and make significant progress towards the termination problem.

3.1 Mean Payoff Algorithm

The first algorithm implemented is referred to in [6] as `Qual-MP`. This algorithm solves the *qualitative* mean-payoff problem for finite state MDPs with rewards. The target of this algorithm is to find the set of vertices such that the value of the mean-payoff objective is satisfied with probability 1 (as is implied by the definition of a qualitative solution). For a given state s in the MDP, the algorithm first finds if there exists a strategy such that the expected mean payoff of the strategy starting from state s is below 0. After that, if applies this strategy onto the MDP, resulting in a Markov Chain. This Markov Chain has the remarkable property that there is always going to be a set of states C such that any sequence of vertices starting from the states s that enters C is almost surely going have a long-run average payoff less than or equal to 0. These set of states C are also called the bottom strongly connected components, or the sink states, of the Markov Chain. It is clear from here then, that any strategy τ that tries to solve the problem of minimizing the mean payoff must try to maximize the probability of reaching these vertices in the MDP (this can by solved by the method `Max-Reach`). Furthermore, given such a strategy τ, we are only interested in the states such that τ ensures that the probability of reaching that state u in the given strategy is 1. These sets A are those which it is guaranteed that the probability that the mean payoff is less than or equal to 1 almost surely.

Algorithm 1: `Qual-MP` Pseudo Code, algorithm taken from [6]

```
Data: Finite State MDP $\mathcal{D}$, Reward function $r : V \to \{0,1,-1\}$
Result: The set of of vertices such that the value of the minimizing mean payoff
        objective is equal to 1 almost surely.
1  $V_? = V, A = \emptyset, T = \emptyset, \hat{r} = r$
2  while $V_? \neq \emptyset$ do
3      $s \longleftarrow$ Extract($V_?$)
4      if $\exists$ a strategy $\rho$ where the mean payoff starting from that state is less than 0
         for MDP $\mathcal{D}$ with reward structure $\hat{r}$ then
5          $\rho \longleftarrow$ get-MD-min($\mathcal{D}, r, s$)
6          $D(\rho) \longleftarrow$ a Markov Chain, fixed by strategy $\rho$
7          $C \longleftarrow$ a BSCC of $D(\rho)$ that is not in $A$ such that the probability of
             reaching this set of states $C$ is 1
8          $(\tau, (reach_v)_{v \in V}) \longleftarrow$ Max-Reach($\mathcal{D}, C \cup A$)
9          $A' \longleftarrow \{u \in V \mid reach_u = 1\}$
10         add $A'$ to $A$
11         for every $u \in V$ do
12             if $u \in A$ then
13                 $\hat{r}(u) \longleftarrow 0$
14         end
15         if $s \notin A$ then
16             add $s$ to $V_?$
17 end
18 return $A$
```

In Algorithm 1, $V_?$ represents the unexplored vertices, A is the set of vertices that we are looking for, and $\hat{r}$ is a modified reward function which starts off identical to the MDPs reward function, but then changes through modifications in line 13. The algorithm also has several additional methods that we must implement. `Extract` is simply a random removal of a vertex from the set of unexplored states $V_?$, `get-MD-min` takes the original MDP $\mathcal{D}$ with reward structure r and a starting state s and tries to find the optimal policy (or set of actions) that minimizes the mean payoff starting from state s. Finally, `Max-Reach` solves the reachability objective which amounts finding the optimal policy that maximizes the probability of reaching a vertex in $C \cup A$. Furthermore, $(reach_v)_{v \in V}$ denotes the reachability probabilities that `Max-Reach` returns for the optimal strategy τ that it computes.

3.1.1 Theoretical Complexity: `Qual-MP`

Outside of `get-MD-min`, and `Max-Reach`, the vast majority of the operations in this algorithm are either reassignments of states, or finding bottom strongly connected components. It is well known that finding the BSCCs of a Markov Chain are easily doable in polynomial time, either through techniques such as Depth First Search or calculating the steady state probabilities. Thus if `get-MD-min` and `Max-Reach` run in polynomial time, then `Qual-MP` clearly should run in polynomial time. Both aforementioned procedures, however, have provably amounted to simple Linear Programming problems, as proposed in [13] that can be solved in polynomial time through the interior point method, implying that the average case runtime of these algorithms, and by extension `Qual-MP`, is polynomial as well.

3.2 Qualitative Cover Negative Algorithm

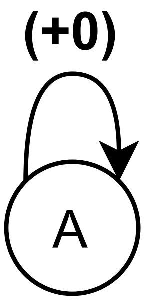

Figure 3.1: Example of an MDP that satisfies the Mean Payoff Objective but not the Cover Negative

The Cover Negative and Mean Payoff problems, albeit similar, are not entirely identical. That is, the set of strategies Σ_{CN} that solve the cover negative problem actually subsume the set of strategies Σ_{MP}, and so there exists MDPs and associated strategies that can solve the Mean Payoff but don't necessarily solve the Cover Negative. A good example is the MDP in Figure 3.1. The long-run average reward here evidently is equal to 0, satisfying the mean payoff objective that `Qual-MP` seeks to solve. However, it fails to satisfy the cover negative objective as the counter value (or the value of the reward) doesn't cover arbitrarily low values in $\mathbb{Z}$. Thus, we need to make modifications to any MDP so as to ensure that $\Sigma_{CN} = \Sigma_{MP}$. To do this, we must convert an arbitrary MDP into a *decreasing* MDP. A decreasing MDP $\mathcal{D}$ has the property that any memory less deterministic strategy in $\mathcal{D}$ starting from any state u has an associated finite path w such that the total reward gained at the end of $w = -1$ [6]. This ensures that counter value (or total reward value) is able to cover arbitrarily low negative values and thus by extension, solve the Cover Negative problem.

Now, to solve the Quantitative Cover Negative Problem (i.e. get the maximum probabilities that cover negative is satisfied starting from each vertex), we must first solve the qualitative problem, i.e. find the set $A \subseteq V$ of vertices such that the maximum probability of satisfying the Cover Negative ($LimInf(= -\infty)$) objective is equal to 1. On paper this algorithm looks simple, and is shown in Algorithm 2

Algorithm 2: `Qual-CN` Pseudo Code, algorithm taken from [6]

```
Data: Finite State MDP $\mathcal{D}$, Reward function $r : V \to \{0,1,-1\}$
Result: The set of of vertices such that the value of the cover negative is equal to
        1 almost surely.
1 $\mathcal{D}' \longleftarrow$ Decreasing($\mathcal{D}$)
2 $A' \longleftarrow$ Qual-MP($\mathcal{D}', r$)
3 $A \longleftarrow \{v \in V \mid (v,1,0) \in A'\}$
4 return $A$;
```

Evidently we see in Algorithm 2 that the most important part of the procedure is the conversion on the original MDP $\mathcal{D}$ into the decreasing MDP $\mathcal{D}'$ discussed earlier. From a cost perspective, this algorithm requires a polynomial time algorithm from before (`Qual-MP`) and a new algorithm `Decreasing`, which allows us to equate the Mean Payoff problem to Cover Negative. We now explain this procedure.

3.2.1 Creating a Decreasing MDP

Earlier in this section we explained why exactly we want to create a decreasing MDP. Now we discuss the mechanics behind creating said decreasing MDP. Given an MDP $\mathcal{D}$, nodes in the new MDP $\mathcal{D}'$ are responsible for storing information about the states, counters, and individual transitions in $\mathcal{D}$. In a decreasing MDP, we are essentially going through every possible transition in the MDP and tracking some important information about the runs at each node. Every node in the new MDP is a list containing three or four items, so every vertex is either of the form (u,n,m), or $[u,n,m,v]$. Furthermore, vertices of the form $(u,1,0)$ for some vertex u in $\mathcal{D}$ are known as checkpoints. For each of the vertices in $\mathcal{D}'$, the coordinates have the following meanings:

- **First Coordinate** (u): the current vertex in $\mathcal{D}$
- **Second Coordinate** (n): the number by which the counter needs to be decreased to make the sum of rewards since the last checkpoint
- **Third Coordinate** (m): the number of steps since the last checkpoint
- **Fourth Coordinate** (v): *(if present)* the next vertex of $\mathcal{D}$ through which a run with decreasing counter value should continue.

Formally, we can define the decreasing MDP as $\mathcal{D}' = (S', \hookrightarrow, S'_N, S'_P, Prob')$, where the following is true [6]:

1. $S' = \{(u,n,m),[u,n,m,v] \mid u \in S, u \to S, 0 \le n,m \le |S|^2+1\} \cup \{div\}$
2. $S_P = \{[u,n,m,v] \in S' | s \in S_p\}$
3. $S_N = S \setminus S_p$
4. $\hookrightarrow$ is the transition relation defined by the least set such that for every $u,v \in S$ such that $u \hookrightarrow v$ and $0 \le n,m \le |S|^2+1$:
   - $m = |S|^2+1, n > 0 \implies (u,n,m) \hookrightarrow div$
   - $m \le |S|^2+1, n = 0 \implies (u,n,m) \hookrightarrow [u,1,0,v]$

- $m < |S|^2 + 1, n > 0 \implies (u,n,m) \hookrightarrow [u,n,m,v]$
- if $u \in S_P, [u,n,m,v] \hookrightarrow (v, n + r(u), m+1)$ and $[u,n,m,v'] \hookrightarrow (v,1,0)$ for all $v' \neq v \in S$ such that $[u,n,m,v] \in S'$
- if $u \in S_N, [u,n,m,v] \hookrightarrow (v, n + r(u), m+1)$

$\mathcal{D}'$ has an auxiliary state *div*. *div* is an absorbing state which "punishes" the agent for picking a bad action/going along a bad transition through which the Mean Payoff objective is not able to be satisfied. Thus, once the player gets stuck in *div*, there is **no way** that the Mean Payoff objective can be satisfied. When any run in this MDP starts at a checkpoint, the second counter starts with the value 1 because we are waiting from the sum of rewards to be -1 and the counter value is initially set at value 0. As the play proceeds and different states have different rewards, the counters get updated accordingly. The moment counter value 0 is reached, we reach a checkpoint and the counter values are reset to 1 and 0 respectively. As shown in [6] we can provably bound n and m by $|V|^2 + 1$. Decreasing MDPs are also central to the termination objective that this project revolves around, because as proven in Lemma A.2 in [5], decreasing MDPs are constructed in such a way that preserves the property of optimal termination probability being equal to 1.

Another important aspect of this MDP is the new probability and reward matrices. For any probabilistic state $[u,n,m,v]$ we say that the probability of the transition $[u,n,m,v] \hookrightarrow (v',n',m') = P(u \rightarrow v')$ in the original MDP $\mathcal{D}$. Also the reward at every 3 element state $(u,n,m) = 0$, at $[u,n,m,v] = r(u)$, where $r(u)$ is the reward in the original MDP of being at the state u and *div* is a sink state with reward 1.

3.2.1.1 Theoretical Complexity: `Decreasing`

A careful look at the exact procedure for creating a decreasing MDP reveals that the procedure involves travelling across possible runs in the MDP and keeping a track of certain variables along the way. Given that the MDP is equally represented as a set of directed graphs, it is easy to see that this algorithm revolves around a slightly modified implementation of a Depth First Search algorithm on the **modified** set of vertices, which will run in polynomial time on the new state space. Since these new vertices are keeping track of more information than the previous MDP, there is a significant blowup in the size of the state space. However, this blowup is still polynomial, in the order of approximately $O(n^3)$, and thus the algorithm still technically runs in polynomial time.

3.2.2 Theoretical Complexity: `Qual-CN`

In the previous section, we explained why `Decreasing` will run in polynomial time, and we also explain earlier why `Qual-MP` also runs in polynomial time. The rest of the procedure simply involves iterating through the output of `Qual-MP` on $\mathcal{D}'$ and extracting the checkpoints. This can be done fairly efficiently using sorting algorithms that run in $O(nlogn)$, which doesn't change the overall polynomial complexity of the algorithm which will still run in polynomial time.

3.3 Quantitative Cover Negative Algorithm

Once the qualitative cover negative problem has been solved, i.e. we know which vertices in $\mathcal{D}$ have the property that $P(LimInf = -\infty) = 1$, then solving the quantitative cover negative problem, i.e. getting the probabilities of achieving cover negative at each vertex boils down to simply maximizing the probability of reaching the subset of vertices A that we computed in Algorithm 2. The algorithm for `Solve-CN` thus is straightforward, and is shown below:

Algorithm 3: `Solve-CN` Pseudo Code, algorithm taken from [6]

```
Data: Finite State MDP $\mathcal{D}$, Reward function $r : V \to \{0,1,-1\}$
Result: The probabilities of satisfying the cover negative objective from each
        vertex.
1 $(A,\tau) \longleftarrow$ Qual-CN($\mathcal{D}, r$)
2 $(val_v)_{v\in V} \longleftarrow$ Max-Reach($\mathcal{D}, A$)
3 return $(val_v)_{v\in V}$;
```

`Solve-CN` uses methods that we have previously explained run in polynomial time, and therefore, we can also say that `Solve-CN` runs in polynomial time.

3.4 Finding a Bound for the Termination Value of the Game

A valid question to ask at this stage is how the above algorithms are related to our ultimate objective of implementing algorithms that approximate the termination value of a Markov Decision Process. The answer to this lies in the reductions made in [5].

3.4.1 Motivation

Denote the termination objective given a certain start state q and counter value i by the combination $Term(q,i)$. Then we can say that for any fixed state q and counter values $i \leq j$,

$$\mathrm{Val}(Term(q,i)) \geq \mathrm{Val}(Term(q,j)) \geq 0$$

This makes intuitive sense because the higher up a counter value we go, the less likely it is that we reach counter value 0, hence why the game values for lower counter values are higher. So we have a monotonically decreasing sequence of game values, and so if we define $\mu_q := \lim_{i\to\infty} \mathrm{Val}(Term(q,i))$, then we can find an arbitrarily large i such that $\mu_q \leq \mathrm{Val}(Term(q,i))$, which amounts to being able to decrease the counter by an arbitrary value with probability at least μ_q. This amounts to the Cover Negative Objective, and the associated value v_q can be defined as follows:

$$v_q := \mathrm{Val}(LimInf(= -\infty), q)$$

where the additional q simply symbolizes the value of Cover Negative given that we start from state q. One interesting point to note here is that while we start from the termination problem which heavily depends on the initial counter value, we gradually reduce the problem into one that is independent of counter value which reduced the difficulty of implementation significantly. Most importantly, [5] also show that $\mu_q = v_q$.

It can also be shown there there exists a large enough N such that for all states q and counter values $i \geq N$,

$$\text{Val}(Term(q,i)) - \mu_q \leq \varepsilon$$

for a given $\varepsilon > 0$. This means that we can use our previous implementations to ε-approximate the termination value of the game for all counter values i above a certain constant N. Computing this N itself, however, is nontrivial and will be the final algorithm we will implement in this project.

3.4.2 Bounds

Let us call the algorithm that computes the bound N as `Bounds`. Now given a *decreasing* MDP $\mathcal{D}'$, Lemma 3.6 in [5] states that given $\mathcal{D}'$, we can construct a system of linear inequalities $\mathcal{L}$ such that there is a tuple of solutions $(\bar{x}, (\bar{z_q})_{q\in Q})$ that solves $\mathcal{L}$ and is polynomial in encoding size of $\mathcal{D}'$. While the specifics of this inequality $\mathcal{L}$ are irrelevant, note that we can get the singular number $\bar{x}$ and the associated vector of values $\bar{z_q}$ by using the optimal values for minimizing the discounted total reward in $\mathcal{D}'$ using a maximum discount factor $\Lambda < 1$ (the paper explains how to find this Λ but because we don't implement algorithms to compute the value of the discounted reward objective, this is irrelevant to the project). After further mathematical manipulations with the values of the discounted total reward we arrive at the vector of optimal solutions $\bar{z_q}$ for each state in the decreasing MDP and the number $\bar{x}$. Given these solutions to the linear program, [5] then show that $N := \max\{h, \log_c(\varepsilon \cdot (1-c))\}$, where

$$c = \exp\left(\frac{-\bar{x}^2}{2 \cdot (z_{\bar{\max}} + \bar{x} + 1)}\right)$$

and $h = \lceil z_{\bar{\max}} \rceil$, where $z_{\bar{\max}}$ is the difference between the maximum and minimum of the z_q of each state q in the MDP. Now computing the exact value of c and h is a slightly technically complex process to implement, and due to time constraints and some difficulties in implementing the total discounted reward objective, we omit implementing an exact version of this algorithm. Rather, through smart approximation, we can nevertheless compute an exponential upper bound on the value N which is still practical.

Through elementary linear programming principles, it can be shown that $c \leq e^{-e^{-p(||\mathcal{D}||)}}$ and $h \leq e^{p(||\mathcal{D}||)}$, where $|| \mathcal{D} ||$ is the bit encoding size of the MDP *before Decreasing*, and p is a positive polynomial. To find the correct polynomial we use a theorem from **insert reference** that for any linear program with n variables, if R is the encoding size of any inequality that forms any one of the constraints of the LP, then if there is a feasible optimal solution to the LP, the rational feasible optimal solutions to the LP require at most $4Rn^2$ bits to encode any rational coefficient in this optimal feasible solution. Now recall that $\mathcal{L}$ is a set of linear inequalities formulated on a decreasing MDP. Thus, we can say that n is the higher number of states in $\mathcal{D}'$, which is a cubic blowup of the number of states in $\mathcal{D}$ (of the order $O(n^3)$), while the largest inequality is simply a linear combination of the states in $\mathcal{D}'$, which means that the size of the inequality has an upper-bound of the encoding size of $\mathcal{D}'$. Hence, by this logic, we can formulate a rough algorithm that approximates the upper bound N, which we will explicitly detail in Section 4.5. Just like all other algorithms implemented, the theoretical complexity of `Bounds` is also polynomial because at the end of the day we have reduced the problem to several simple mathematical computations that will always run in polynomial time.

Chapter 4

Implementation Details

This honours project is concerned with the implementation of the algorithms detailed in Chapter 3. In this Chapter, we will go through the specific problems behind implementing some of these algorithms and how we have dealt with them.

4.1 Overview of Technologies and Algorithms Used

These algorithms have been implemented in Python for several reasons. Firstly, Python is a widely used language that is easy to understand. For the sake of reproducibility of our algorithms, the use of Python allows one to focus on capturing the mathematical subtleties and provides access to a wide range of libraries that one can use. For the analysis of Markov Decision Processes we use the `MDPToolbox` library, which provides a set of functions that solve a range of problems for MDPs [7]. These functions "solve" a Markov Decision Process by finding the optimal deterministic policy for a specific optimization criterion. These optimization criterion define value functions $V^{\pi} : S \to R$ that provides the expected "performance" of a policy at a given state s. The MDP Solver aims to find an optimal policy such that Value Function is *maximized* or *minimized*. The two key objectives that we use are the average infinite reward and the total reward. The former maximizes (or minimize) the long-run average expected reward of the MDP while the latter is focused on maximizing the total reward of the resultant MDP. The latter will prove to be necessary when trying to evaluate reachability objectives, which will be defined in this section.

4.1.1 Defining an MDP

We define an MDP using the formluation described in [7], which is more often used for reinforcement learning and stochastic optimization. Translating it to our game theoretic definitions is, however, trivial. In our implementation, we define an MDP based on its states and actions.

Definition 8. *(Taken from [7]) A MDP is a tuple (S, A, p, r) where*

- *S is the finite set of states describing the possible configurations, or states of the* *system*

- *A is the set of possible actions or decisions controlling the interaction between states*
- *p is a probability state transition function that takes two states $s, s' \in S$ and an action $a' \in A$ and returns the probability $p(s \mid s', a')$ which is the probability of moving from state s' to s by taking action a'.*
- *r is the reward function on the possible outgoing transitions from each state $r(s|s', a')$.*

Notice the difference in definitions of MDPs from those traditionally defined in the game theoretic context previously defined versus this new definition. First, the state space has not been partitioned into control and probabilistic states. Now, every state is a probabilistic and control state at the same time. Second, the reward function is now defined on transitions and actions instead of states. This is not a significant problem, however, because we can just assign the same reward value for all transitions out of the state for a particular action. The main problem is to write methods that are able to convert take an MDP of this form and partition it into probabilistic and control states. We achieve this by modifying our state space to include a track record of actions. For example, if we have an MDP with $n > 0$ states and $m > 0$ actions, then we can create a new MDP with the following state set S_{new}:

$$\{0,(0,0),(0,1),...(0,m-1),1,(1,0),(1,1),...,(1,m-1),...,n,(n,0),...,(n,m-1)\}$$

We can thus partition S_{new} into the control S_c and probabilistic S_p states, where the control states are $\{0,1,...n\}$ and probabilistic states are all the tuples in the new state set. Under each action $0 < k \leq m$, every control state $c \in S_c$ has only one outgoing transition to the probabilistic state (c,k). Now since each probabilistic state is a tuple in the format (state, action), these states only have one nontrivial set of transitions in the probability matrix corresponding to the right action. For the transition matrices that do not correspond to the right action, we just loop the probabilistic state back onto itself with probability 1. To ensure that any iteration algorithm does not map a probabilistic state to the wrong action, we make some modifications to the reward structure of the MDP. The original reward matrix has a structure of $(|S|,|A|)$, mapping each state to the reward of entering the state using that action. The new reward matrix has a similar structure, $(|S_{new}|,|A|)$, and we populate it in the following manner: For every state in S_{new}, if the state is a control state, then we set the reward to 0 for all actions because we want any algorithm to be neutral to entering a control state. If the state is a probabilistic state (c,k), however, then for all actions but k, we set the reward opposite to the objective of the iteration algorithm. That is, if the objective is to maximize the value function, then we set the reward to -1. Otherwise, we set the reward to +1 as a way of disincentivizing the algorithm to choose to map the wrong action to these states. For action k, we set the reward equal to the reward of entering state c using action k in the original MDP. While this is a rather inelegant solution to the problem, we must note that `MDPToolbox` is heavily limited in its functionality and that this is the only way we can ensure that we can correctly implement the algorithms discussed in Chapter 3. Note that this results in a quadratic blowup of states, taking an

MDP with n states and return one with $n(m+1)$ states, while preserving the number of actions.

4.1.2 Relative Value Iteration vs. Modified Policy Iteration

`MDPToolbox` provides a wide variety of algorithms that are crucial to implementing some of the algorithms we previously discussed. In this section we shall look at some of these algorithms, how they operate, and some of their complexities. This will be useful when we empirically analyze our own algorithms which utilize some of these tools. Some of the algorithms we utilize for our implementations include relative value iteration, generic value iteration, and policy iteration.

Relative Value Iteration is useful in our implementation for `Qual-MP`, where we check if there exists a strategy such that the Mean-Payoff starting from the state is less than 0 (check line 2 of Algorithm 1). This algorithm deals with the *infinite-horizon* discounted cost case, which means we are looking at the behaviour of the cost in the long-run, while the discounted cost implies that rewards can be discounted as the number of time steps increase. In our situation, we do not consider the discounted case as it is not relevant to our study. As with generic value iteration, relative value iteration relies on finding fixed points of the Bellman operator. The basic idea of the algorithm is to start with a given error ε, a set of possible values and a start state and dynamically update the value vector and optimal policy along the way until either the change in game value between iterations is less than ε, or the maximum number of iterations has been reached (for further details on the mathematics of the algorithm, refer to [13]).

Note, however, that Relative Value Iteration is not the only algorithm useful for solving average reward problems. `MDPToolbox` also allows for using Modified Policy Iteration for this purpose ([7]). While Policy Iteration has been shown to converge faster than generic value iteration but it must also solve more complex linear programs and require more multiplications per iteration, as shown in [13].

The computational complexity of the infinite-horizon average cost problem has also been analysed in literature. As mentioned in [12], the infinite horizon, average cost deterministic problem been proved to lie in the complexity class NC, and it has been proven before that NC $\subseteq$ P. However, this doesn't imply that the algorithms are efficient. As a matter of fact, a critical flaw behind both of these algorithms is that it has been shown that they both suffer from the "curse of dimensionality", in that the complexity increases exponentially with size, specifically because the algorithm requires evaluating the expectation for every state-action pair. To remedy this, researchers have invented algorithms such as the "Empirical Relative Value Iteration" method which provide a crude estimate of the Bellman Operator which converges at significantly lower iterations with less normalized error (see [8] for more).

4.2 Implementing **`Qual-MP`**

The implementation of Algorithm 1 is fairly straightforward. Consider a random state in the MDP s_r.

First, we use Relative Value Iteration through `MDPToolbox` to get the optimal policy that *minimizes the mean expected payoff* (while the Toolbox usually maximizes the value function, we can instead change maximization to minimization by simply changing the sign of the objective function). One minor drawback with this implementation, although it is the only one out there, is that it can only compute ε-optimal values and strategies. If, however, we set ε very close to 1 (something like 0.99), then we can somewhat say that these strategies are simply optimal. Through Bellman's "principle of optimality", we know that the optimal policy computed for any state is the same for all states, which saves us significant computational cost. Of course, if the value at a certain state (i.e. the actual mean payoff) is not less than zero, this means that there doesn't exist a strategy for the MDP which has a mean payoff less than zero starting from that state. The procedure `get-MD-min` is thus unnecessary, because we have already computed it to check for the strategy ρ.

Next, we take our computed strategy ρ and generate a Markov Chain, $D(\rho)$ upon fixing the strategy on the MDP. This is relatively simple because we simply create a transition matrix consisting of rows for each state which are copied from the policy-suggested action's transition matrix. Thus, if the optimal policy that at state 2 we must choose action 2, then our new probability matrix will replace its third row with the third row from the second transition matrix from the MDP.

After that, we must extract the bottom strongly connected components of this resultant Markov Chain. There are two ways to achieve this. One option is to find strongly connected components using pre-existing algorithms such as Depth First Search, and then verify that the probability of reaching these strongly connected components starting from our state of choice is 1. While this strategy technically works, it is also unnecessarily complicated. Finding all strongly connected components uses Depth First Search, which runs on Big O Complexity of $O(|V|)$, where $|V|$ is the number of states in the Markov Chain. Then we would check for each connected component computed whether the probability of reaching that component in our Markov Chain is one, which for larger state spaces, can require large amounts of multiplication. While the DFS procedure is relatively efficient, we can skip altogether and instead compute the steady-state probabilities of each state in the Markov Chain and from the list of steady state probabilities, verify which ones are nonzero. Methods to calculate these probabilities are discussed in Section 4.2.1. Once we have found the sink states/bottom strongly connected components, then we try to maximize the probability of reaching these states through our method `Max-Reach` which involves recreating a new MDP and solving a different objective. We discuss the implementation of this procedure in Section 4.2.2.

Once we have the reachability probabilities for each state, we extract the states that have a probability greater than ε. While the algorithm in the paper asks to find the states with probability = 1, since our method is not exact in nature (and no Python implementations of an exact method exist), we stick to finding probabilities greater than a given constant. This isn't a major problem because if we set ε to be a number very close to 1, then the method essentially works, although convergence of the iteration method may take longer. This gives us set of states A' which ensure that the mean payoff objective is met starting from our initial state s_r. To show the algorithm that we don't have anymore interest in analyzing the behaviour of these states anymore and so we set their reward to 0, and add s back into the set of unexplored vertices till we encounter a configuration of rewards that puts s in its computed A'.

4.2.1 Computing Steady-State Probabilities

Since Python has no existing implementation, we devote this subsection on discussing the theory of our implementation of computing steady state probabilities. There are multiple ways of doing this, one of which involves calculating a system of equations. This is formalized in [14]: Given a vector of steady state probabilities $\overrightarrow{\pi}$ and a transition matrix P we can provide the following constraints

$$\overrightarrow{\pi} = \overrightarrow{\pi}P, \text{ and } \Sigma_{i=1}^{n}\pi_i = 1$$

and then solve a system of equations with these constraints and use the solution to generate the steady state probabilities. Looking at the first constraint, it is easy to see that we can reformulate the problem into one aimed at finding the eigenvalue of the transition matrix, namely

$$0 = \overrightarrow{\pi}P - \overrightarrow{\pi} \implies \overrightarrow{\pi}(P-I) = \overrightarrow{0},$$

where I is the identity matrix. Hence, this is a simple eigenvalue problem where 1 is the eigenvalue. While a given transition matrix can have multiple eigenvalues, note that it has been proved that every transition matrix will certainly have an eigenvalue of 1 (among others), and that all other eigenvalues will be less than 1. Those eigenvalues, however, have no significant meaning. Hence, our algorithm to find the steady state probabilities is formalized below:

Algorithm 4: Pseudo Code to Compute Steady State Probabilities of Transition Matrix P

```
Data: Transition Matrix $P$
Result: Steady State Vector $\overrightarrow{\pi}$
1 $E, V$ = eig($P^T$)
2 $E' = e \mid e \in E \wedge \mid e - 1 \mid \leq 1 \times 10^{-8}$
3 idx = $E$.index($E'$)
4 $\overrightarrow{\pi}$ = $V$[idx]
5 $\overrightarrow{\pi} = \overrightarrow{\pi} / \Sigma_{i=1}^{n}\pi_i$
6 return $\overrightarrow{\pi}$
```

Note that `eig`(M) is a predefined method which finds the eigenvalues E and eigenvectors V of M. In the algorithm, we provided `eig` with the *transpose* of P because **(why is that)**. Furthermore, the additional property of lists called `index` is simply an inbuilt property of lists which provides the index of E' in the list E. Finally note line 5, where we normalize the steady state probability vector π because the eigenvector doesn't tend to sum to 1. Thus to force it to become probabilities, we simply divide it by its sum.

4.2.2 Implementing `Max-Reach`

There is much literature on the study of reachability, especially in works by Puterman [13]. Puterman suggests a linear programming (LP) approach to `Max-Reach`, as well as a computationally suitable approach using the Bellman Operator (what is known today as Value Iteration). Value Iteration, however, can be very slow in the worst case, requiring exponentially more iterations, and so instead we implement a linear programming solution to this problem. This is achievable in Python using the PuLP Library. PuLP is a high-level modelling Python library that provides users access to commercial mixed integer linear programming solvers such as CPLEX and Gurobi [11]. The solver can solve LPs involving 100s of variables and constraints in a matter of seconds and is useful in our implementation of `Max-Reach`, especially in the face of larger MDPs after the quadratic blowup in states discussed in earlier in this chapter. In order to maximize the probability of reaching a set of target states A, we use what are known as the *Bellman Optimality Equations*, which is an intuitive linear program that resolves the objective of maximizing the probability to reach a target set of vertices A. Consider an MDP $\mathcal{G}$ consisting of a set of vertices $S = \{s_0, s_1, ..., s_n\}$ which is partitioned into mutually exclusive control and probabilistic state sets S_c and S_{prob}, respectively. If we have one LP variable x_i for each vertex $s_i \in S$, we can formulate the following LP:

$$\min \quad \sum_{i=0}^{|S|} x_i$$

such that:

$$\begin{aligned} x_k &= 1 & \forall k \in A \\ x_i &\geq x_j & \forall i \in S_c \wedge v_j \in E(v_i) \\ x_i = \sum_{v_j \in E(v_i)} & q_{v_i}(v_j) \cdot x_j & \forall v_i \in S_{prob} \\ x_i &\geq 0 & \forall i \in \{1, ..., n\} \end{aligned}$$

where $E(v_i)$ denotes the outgoing edges from state v_i and $q_{v_i}(v_j)$ denotes the probability of transitioning from v_i to v_j. This linear program is guaranteed to have an optimal solution and the solution vector $(x_1^*, x_2^*, ..., x_n^*)$ is the vector of optimal reachability probabilities for the player in each state of the game.

4.3 Implementing `Qual-CN`

As can be seen from Algorithm 2, the most important sub-procedure in `Qual-CN` is to implement `Decreasing`, which is meant to create a decreasing MDP that keep tracks of the history of the counter values and other properties of runs in the MDP. The original plan to create this decreasing MDP was create two procedure: a `4-Element` and `5-Element` procedure corresponding to tuples and lists respectively in the definition of $\mathcal{D}'$ in Section 3.2.1. From there, we wished to use breadth first traversal to dynamically add nodes and transitions for the directed graph corresponding to each action of the MDP. While such a plan is compatible with `MDPToolbox`, note that it is not compatible with the way we have defined MDPs earlier in this chapter. Since that is the only way we can ensure that `MDPToolbox` is able to incorporate control and probabilistic states, thus implementing `Decreasing` is a far too complex procedure that can be accomplished in such a short space of time using the current Python Library. Thus, for this project we will be unable to implement `Decreasing`, and by extension, `Qual-CN`.

Instead, what we propose is using `Qual-MP` to indirectly solve the Cover Negative problem (qualitatively, of course). As discussed in Sec 3.2 there is a specific subclass of MDPs for which strategies that satisfy the Mean Payoff Objective do not satisfy the Cover Negative Objective. This is because the Mean Payoff only looks for states which will have non-positive average rewards while Cover Negative is looking for states which will almost surely encounter arbitrarily low counter values. So, one way to solve this problem is to strengthen the inequality constraint we have in line 4 of Algorithm 1. Instead of checking for a strategy with mean payoff being less than or equal to 0 from a random state, we can instead check for a strategy with mean payoff being *stricly less* than 0 from a random state. In the case of MDPs like that of Figure **??**, this modified algorithm will return "A" as its average reward is simply 0. In order to test the validity of this, we will run this modified version of `Qual-MP` on a well known subclass of games represented as MDPs.

4.4 Implementing `Solve-CN`

Implementing `Solve-CN` is fairly straightforward. Since we haven't been able to implement `Qual-CN`, we simply execute `Qual-MP` first to find the set A of states that have almost surely will have negative average rewards (over an infinite time horizon) and then try to find the optimal policy that maximizes the probability of reaching A using `Max-Reach`. We will test the accuracy of this modified method in Section 5.3.

4.5 Implementing `Bounds`

To implement the algorithm that calculates the bound for counter value above which the termination values of the game are equivalent to those for the Cover Negative problem, we simply implement the mathematical expressions that were discussed in Section 3.4.2. Ideally the approach needed to get the exact value of the bounds would be to implement the linear programs discussed in [5] and try to minimize the total discounted reward problem for a fixed discount λ in the game, but we decide to opt for a simpler approach of computing the upper bounds on the bound itself for sake of simplicity. For an MDP $\mathcal{D}$ and an error value ε, it can be shown that the upper bound on $N(\varepsilon)$ is equivalent to the following expression:

$$N(\varepsilon) \leq 1 + \exp(p(||\mathcal{D}||)) \cdot \ln(\varepsilon^{-1}) + (1 + p(||\mathcal{D}||)) \cdot \exp(p(||\mathcal{D}||))$$

Where $p(\cdot)$ is a polynomial upper bound on the solution of a linear program. Finding this polynomial is easier due to to a theorem from [1]

Theorem 1. *Consider an LP with n variables. If R is the maximum bit encoding size of any inequality that forms any one of the constraints of the LP, then if there is as an optimal solution to the LP, then there is a rational optimal feasible solution to the LP such that the maximum number of bits needed to encode any rational coefficient in the n-dimension vector of the optimal feasible solution is* $4Rn^2$.

To better understand how we can use this to code `Bounds` let us look at the LP in question, that was described in [5]:

$$\begin{aligned} z_q &\leq -x + k + z_r && \forall q \in S_c \wedge (q,k,r) \in \Delta \\ z_q &\leq -x + \sum_{(q,k,r)\in\Delta} \mathbb{P}(q,k,r)\cdot(k+z_r) && \forall z_q \in S_{prob} \\ x &> 0 \end{aligned}$$

Looking at the second inequality which is basically a dot product of probabilities with the vector of states, we can see there that the maximum size of this inequality would be the number of probabilistic states in the MDP, which in the worst case would be at least equal to the number of control states in the MDP based on the definition we have created in Section 4.1.1. Thus an upper bound on the maximum bit encoding size of an inequality can be $R < ||\mathcal{D}||$, and thus our polynomial is evidently $p(||\mathcal{D}||) = 4||\mathcal{D}||n^2$, thus $N(\varepsilon)$ has been implemented as:

$$N(\varepsilon) \leq 1 + \exp(4||\mathcal{D}||n^2)\cdot\ln(\varepsilon^{-1}) + (1+4||\mathcal{D}||n^2)\cdot\exp(4||\mathcal{D}||n^2)$$

Implementing this is trivial through Python's math operators and the `getsizeof` method which returns the size of any object in bytes (which is $||\mathcal{D}||$). More interestingly, we will investigate the effectiveness of this bound in Section 5.2.

Chapter 5

Experiments

5.1 Solvency Games

To illustrate the applications of the code implemented thus far and to experiment with some of the algorithms implemented, we use a modified version of a well known model of MDPs with widespread applications in mathematical finance known as a *solvency game*. Solvency games are an extension of the classic *gambler's ruin* problem. The gambler's ruin problem is defined as follows: consider a gambler who starts gambling with some initial wealth i and then proceeds to make a sequence of bets. For each bet the gambler wins with probability p and loses \$1 with probability $1 - p$. The gambler must quit when (s)he is broke [10] (wealth of 0). Plenty of analysis has been done into this subfield of Markov Chains, specifically in optimal strategies and the expected time taken for the player to go broke. Berger and Vazirani introduce an extension of this model to incorporate Markov Decision Processes in something known as *solvency games*. At every round of this game, the player has the ability to chose between playing two different games each with different payoffs and chances of winning. While they suggest using an infinite state MDP, with states corresponding to the gambler's wealth at each point, we can represent an infinite state MDP as a finite state OC-MDP, and represent the wealth gained of the player at a certain time t as the total reward gained at that time step. Such a style has been used before when investigating solvency games with discounted reward structures [3]. Thus, we propose a slightly unique structure of solvency games. As shown in Figure 5.1, the MDP we experiment with has 1 control state (Play) represented by the blue box, and probabilistic states simulating the Games, Outcomes, cashing-in, and one auxiliary state to increase the payoff of winning Game A.

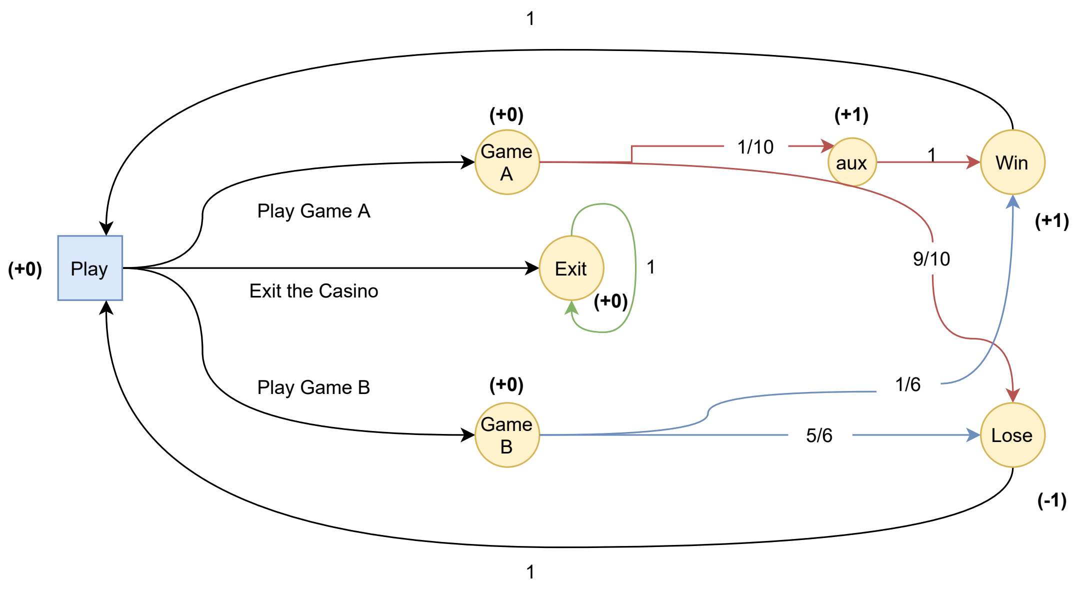

Figure 5.1: Variation on the standard model of solvency games that we use for our experiments.

5.2 Evaluating the Convergence of Termination Probabilities for varying values of N

5.2.1 Motivation

Consider the MDP from Figure 5.1. Regardless of the value of ε, notice that $n = 7$, and even when we ignore the encoding size of n, this means that $N(\varepsilon)$ will be impossible to calculate as $4n^2$ alone is equal to 196 which is far too high an index to raise e to the power of. Thus, we must come up with a heuristic, also known as a finite state "approximation", to empirically calculate the termination probabilities that is heavily inspired from [9].

5.2.2 Methodology (Finite State Approximation)

Recall from Chapter 2 that termination probabilities are expressed in terms of configurations of the form (s, j) where $s \in S_c$ is a control state and $j \in \mathbb{Z}$ is a counter value. Note that the idea of a termination objective is to compute the maximal probability of getting the counter value to go below 0 starting from a given configuration (s, j). According to theory presented in [5], this can happen in one of two ways: either the counter value hits 0 by enough transitions with rewards of -1, or we hit a counter value N above which the probabilities are already defined through `Solve-CN` for each state s. Thus, to maximize the termination probability for any configuration (s, j) we create a new MDP which provides the option of terminating in both ways and then implementing `Max-Reach` which will either try to get the counter value below 0 or reach N, whichever is closer. Formally, we achieve this through the following:

Given an MDP $\mathcal{D}$ with probabilistic and control state sets S_p and S_c respectively and a transition set Δ. We can create a new MDP $\mathcal{G}$ by using configurations as states in $\mathcal{G}$. For a fixed value N we define a new MDP $\mathcal{D}'$ that simulates finding the termination value for an MDP with states as configurations of the form (s, j) for each control state $s \in S_c$ and $j \in \{0, ..., N\}$. Formally this MDP is defined as follows:

$\mathcal{D}' = (S', \hookrightarrow, (S'_c, S'_p), Prob', N)$ where:

- $S'_c := (s, j)$ such that $s \in S_c$ and $j \in [0, N]$
- $S'_p := (s, j)$ such that $s \in S_p$ and $j \in [0, N] \cup$ "TARGET"
- $S' := S'_c \cup S'_p$

The transition relation $\hookrightarrow$ is defined as follows:

- $\forall (s, j)$ such that $s \in S_c \wedge (j = 0 \vee j = N)$,
  - $(s, 0) \hookrightarrow$ "TARGET" if $\exists v : s \hookrightarrow v$ and $r(v) = -1$
  - $(s, N-1) \hookrightarrow$ "TARGET" if $\exists v : s \hookrightarrow v$ and $r(v) = 1$
- $\forall (s, j)$ such that $s \in S \wedge j \in [0, N)$
  - $\forall s \in S_c, (s, j) \hookrightarrow (v, j + r(v))\ \forall s \hookrightarrow v$ and $r(v) > 0$
  - $\forall s \in S_p$, rules above for $\hookrightarrow$ still apply, and also, $Prob'((s, j) \hookrightarrow (v, j + r(v))) = Prob(s \hookrightarrow v)$

Note that we did not define the reward structure of $\mathcal{D}'$ as it is irrelevant to our study. In order to find a suitable bound N for the problem such that for all counter values $j \geq N$, $|\text{Val}(Term, (s, j)) - v_s| < \varepsilon$, we create the MDP just described for increasing values of N until we see the termination probabilities for the configurations beginning to converge, as at that point we would have already visited the correct bound N for this particular MDP and error ε. Recall from Section 4.5 that the key problem of exponential bound $N(\varepsilon)$ was that the index was far too high. Thus, for $\varepsilon = 0.001$, we experiment with potential values of N following an exponential pattern and checking for a value a coutner value j such that the variation in the termination probabilities for control state "PLAY" for higher and higher counter values j is less than ε, because by definition of an ε-optimal value, this will mean that we will have found said value. According to the formula for $N(\varepsilon)$ in Section 4.5, this should be an incredibly high number and we verify this through our experiment.

5.2.3 Results

Table 5.1: ε-Optimal Probability of Termination for ("PLAY" ,1) after constructing an MDP $\mathcal{D}'$ from the modified solvency game and cutting off configurations above N

N	1	2	7	20	54	148	203
$\mathbb{P}(Term)$	0.565	0.652	0.825	0.909	0.981	0.991	0.99

For the purposes of this study, we chose to experiment with values of $N = \lfloor \exp(x) \rfloor$ for

$x \in \{0, 1, 2, ..., \}$ till we notice convergence in the probability of termination starting from control state "PLAY" with counter value 0 for increasingly higher counter values till convergence is achieved (i.e. variation in ε-optimal probability is less then ε). This means that we cutoff the MDP for values $N \in \{1, 2, 7, 20, 54, ...\}$. Figure 5.2 shows the change in the ε-Optimal Probability of Termination starting from "PLAY" with counter value 1 when we cut off the MDP for varying values of N, and Table 5.1 shows the ε-optimal probabilities of termination themselves for varying values of N.

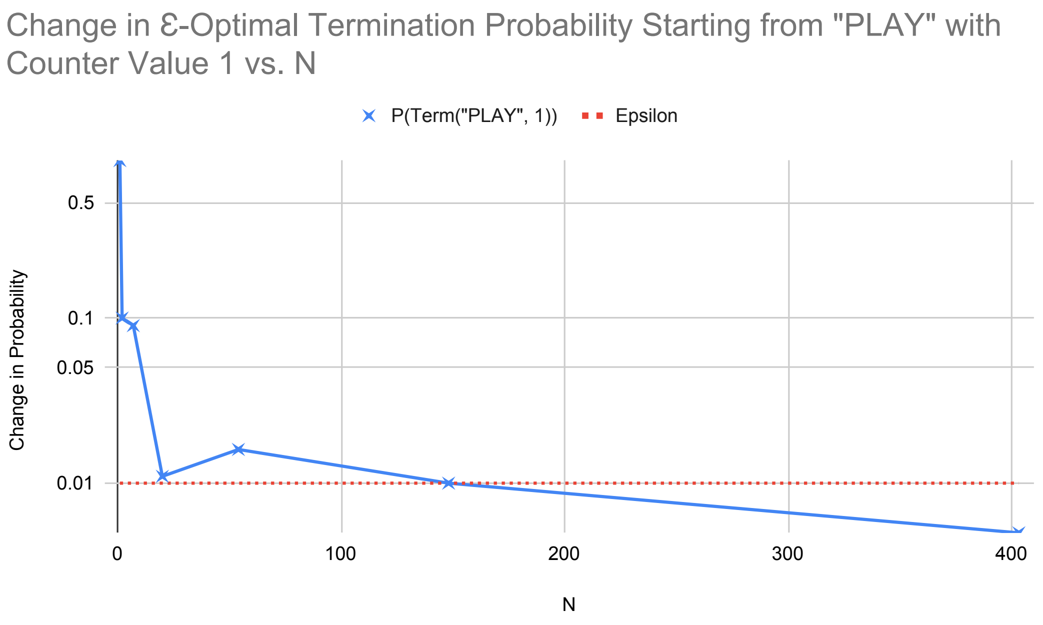

Figure 5.2: ε-Optimal Probability of Termination vs. N

The results from Figure 5.2 and Table 5.1 show something rather interesting. We see that after bounding $\mathcal{D}'$ with counter value $N = e^6 = 403$, the variation in the ε-optimal probability of termination rapidly decreases below $\varepsilon = 0.01$, indicating that convergence has been achieved. This means that by the time the player's wealth has increased by approximately \$403, they are able to play the cover negative strategy, implying that the ε-optimal probability of termination is equivalent to the probability of starting from "PLAY" from any total wealth and cover arbitrarily low counter values (this is the "Cover Negative" Objective we have referred to earlier), and the probability of termination is equivalent to that of Cover Negative being achieved. Compared to the bound we devised in Section 4.5 this is drastically low implying that further investigation needs to be done into tightening the bound or implementing the appropriate linear programming tools to solve the large system of inequalities discussed in [5] to find the appropriate bound for N.

5.3 Using the Mean Payoff problem to solve the Quantitative Cover Negative Problem

5.3.1 Motivation

Recall from Section 3.2 that the set of strategies that satisfy the mean payoff problem are not equivalent to the set of strategies that satisfy the cover negative problem because there exist situations where an MDP satisfies the Mean Payoff objective even though it doesn't satisfy Cover Negative (an example is given in the relevant Section). That situation, however, is unlikely to occur in usual scenarios. If in the general case it turns out that the probabilities of the mean payoff problem are accurate, then that reduces the difficulty of this problem significantly as `Decreasing` results in a quadratic blowup of states which can cause significant memory problems for larger MDPs.

5.3.2 Methodology

In order to evaluate the accuracy of using Mean Payoff instead of `Qual-CN` for our problem, we reconsider the modified version of the solvency games. Note that because we were unable to implement `Qual-CN` we decided to implement `Solve-CN` by first finding the states that satisfy the Mean Payoff problem with probability 1 and then trying to maximize the probability of reaching those states through `Max-Reach`. If we refer to Table 5.1, we see that the termination probabilities of the game are starting to tend towards 1. Since [5] stipulates that for values above $N \geq e^6$, the ε-optimal probabilities of satisfying the termination objective are equivalent to those of the Cover Negative objective, this should mean that the values of the Cover Negative game should be 1.

5.3.3 Results

Indeed, when we run our modified version of `Solve-CN` on the original MDP $\mathcal{D}$, we find that all of the states of the Solvency Game apart from "EXIT" are guaranteed to satisfy the Mean Payoff outcome with probability 1 (that is, the average payoff per turn on each state is guaranteed to be strictly lower than 0). Implementing `Max-Reach` on this, of course, is trivial, as it tells us that each state (except "EXIT") is guaranteed to reach the subset consisting of itself with probability 1 (note that our converged termination probabilities are within $\varepsilon = 0.01$ of this, which further validates our methodology). Furthermore, an interesting aspect to note is the lowered computational cost of running this modified version of `Qual-CN` has on the MDP. Note that due to the number of states here with net reward 0, imposing the strict requirement to have a negative mean payoff meant that after the first iteration where the algorithm classed all states but "EXIT" in the output set A, the rest of the iterations simply involved a trivial elimination of the unexplored vertices as the new reward matrix was full of zeroes, implying that the mean reward would have been 0 for every single state. If the requirement in Line 4 of Algorithm 1 required the mean payoff to be non-positive, then would have required creating linear programs and computing steady state probabilities for 6 more iterations. More importantly, the computed set A would also include the state "EXIT"" which is not consistent with the termination problem because there is no way for the player to start with Wealth 1, Exit the Game, and go broke. Thus, our implemented algorithm for `Solve-CN` which uses a slightly modified version of `Qual-MP` is good enough to approximate the optimal termination probability for a high enough counter value for any commonly occurring MDP.

Chapter 6

Conclusion

This project offers the first known implementation of One Counter Markov Decision Processes (OC-MDPs), and is the first to implement algorithms in Python that can provide ε-optimal approximations of the termination value (ε-optimal termination probability) of OC-MDPs. While the original objective was to implement algorithms that can compute the termination probabilities for a given MDP and error ε for **all** counter values and control states, due to the lack of fundamental methods needed to solve this problem, we have had to develop most tools from ground up which has made the problem significantly more complicated than previously expected. This includes offering the first known implementation of a One-Counter Markov Decision Process in Python, computing steady state probabilities for a Markov Chain, solving the reachability problem for generic MDPs, and developing the first implementation for solving the Qualitative Mean Payoff problem[1] (see Chapter 4 for full details). Thus we have computed the ε-optimal termination probabilities for *most* MDPs (with the exception of a specific type of MDP) and any configuration with a counter value above a number N. Since the method for computing this bound N uses algorithms that are currently not supported by the leading library for MDP analysis, we instead implement an approximation from [5] that provides a crude upper bound for this counter value N using linear programming concepts. For counter values lower than N, we implement a heuristic known as a "finite state approximation" that borrows techniques used to analyze Quasi Birth Death Processes (this is formally defined in Section 5.2.2). In Chapter 5, we justify the minimal impact of the limited scope of our project and our heuristic for termination by experimenting our code on a known subclass of OC-MDPs known as Solvency Games which has applications to mathematical finance and modelling the behaviour of risk-averse investors. Our results show that crude upper bound that we devised in Section 4.5 is not good enough according to our finite state approximation and that further work must be done to implement the *exact* value N for a given error ε and finite state MDP. Furthermore, we notice that the polynomial blowup of states to create the a new MDP that tracks certain properties detailed in Section 3.2.1 is not necessary for the average MDP (such as Solvency Games).

[1]Remember that this requires finding the set of vertices in the MDP for which the optimal probability for the agent to have a non-positive average reward is 1.

Nevertheless, this project leaves several open problems. An obvious open investigation (and continuation of this project) is to successfully convert an MDP into a *decreasing* MDP where every state has a finite path such that the total reward across that path is at least -1. The intention behind this is to ensure that `Solve-CN` can be used on any MDP to find the set of states that almost surely satisfy the Cover Negative Objective for any initial counter value. Implementing and experimenting with some of the bounds on the length of the finite path is another potential avenue for investigation. Furthermore, as the experiments in Section 5.3.3 have shown, for a given error ε, the MDP's bound $N(\varepsilon)$ such that the value of the termination and cover negative objectives are equivalent is significantly lower compared to the upper bound computed in the Appendix of [5]. Thus, a potential extension of this project is to implement the required algorithms using the discounted reward objective to solve the LPs, get the exact solution, and using that to get the exact bound $N(\varepsilon)$. Alternatively, further research can be done into exploring if the polynomial used as an upper-bound for the encoding size of a solution to the Linear Program can be lowered to dramatically lower the size of $N(\varepsilon)$ and improve the bound without having to implement algorithms for discounted total reward. Finally, to complete implementation of the approximation algorithm, it is important to implement the correct algorithm detailed in [5] to compute the termination values of the game for configurations lower than counter value N. The algorithm requires constructing a MDP $\mathcal{D}'$ slightly similar to what we have made for our Finite State Approximation; however, there are some key modifications that could have been made, time permitting.

Bibliography

[1] Computational complexity. *Linear Programming and its Applications*, page 31–44.

[2] N. Berger, Nevin Kapur, L. Schulman, and V. Vazirani. Solvency Games. *Electron. Colloquium Comput. Complex.*, 2008.

[3] Tomas Brazdil, Taolue Chen, Vojtěch Forejt, Petr Novotný, and Aistis Simaitis. Solvency markov decision processes with interest. *Foundations of Software Technology and Theoretical Computer Science*, 2013.

[4] Tomáš Brázdil, Václav Brožek, and Kousha Etessami. One-Counter Stochastic Games. *arXiv:1009.5636 [cs]*, September 2010. arXiv: 1009.5636.

[5] Tomáš Brázdil, Václav Brožek, Kousha Etessami, and Antonín Kučera. Approximating the Termination Value of One-Counter MDPs and Stochastic Games. *arXiv:1104.4978 [cs]*, July 2011. arXiv: 1104.4978.

[6] Tomáš Brázdil, Václav Brožek, Kousha Etessami, Antonín Kučera, and Dominik Wojtczak. One-Counter Markov Decision Processes. *arXiv:0904.2511 [cs]*, September 2009. arXiv: 0904.2511.

[7] Iadine Chadès, Guillaume Chapron, Marie-Josée Cros, Frédérick Garcia, and Régis Sabbadin. MDPtoolbox: a multi-platform toolbox to solve stochastic dynamic programming problems. *Ecography*, 37(9):916–920, 2014. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/ecog.00888.

[8] Abhishek Gupta, Rahul Jain, and Peter W. Glynn. An empirical algorithm for relative value iteration for average-cost mdps. *2015 54th IEEE Conference on Decision and Control (CDC)*, 2015.

[9] J. Lambert, B. Van Houdt, and C. Blondia. A policy iteration algorithm for markov decision processes skip-free in one direction. *Proceedings of the 2nd International ICST Conference on Performance Evaluation Methodologies and Tools*, 2007.

[10] Tom Leighton and Ronitt Rubinfeld. Lecture notes in mathematics for computer science, December 2006.

[11] Stuart Mitchell, Stuart Mitchell Consulting, and Iain Dunning. Pulp: A linear programming toolkit for python, 2011.

[12] Christos H Papadimitrou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of Operations Research*, 12(3), Aug 1987.

[13] Martin L. Puterman. The average reward criterion - multichain and communicating models. In *Markov Decision Processes*, pages 441–491. John Wiley & Sons, Ltd, 1994. Section: 9 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470316887.ch9.

[14] Hamdy A. Taha. *Operations research: an introduction*. Pearson/Prentice Hall, 2011.